VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Rozhodovací stromy a jejich aplikace v analýze obrazu

Decision Trees and their Applications in Image Analysis

2016 Bc. Helena Nedošínská

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Helena Nedošínská**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Rozhodovací stromy a jejich aplikace v analýze obrazu
Decision Trees and their Applications in Image Analysis

Jazyk vypracování: čeština

Zásady pro vypracování:

Analýza obrazu může být založena na využití pohybujícího se okna, na výpočtu příznaků a konečně na použití klasifikátoru. Častěji používanými příznaky jsou například histogram orientovaných gradientů (HOG) nebo příznaky Haarovy. Oblíbenými klasifikátory jsou například SVM klasifikátory (support vector machine), rozhodovací stromy nebo lesy rozhodovacích stromů (random forests). Cílem diplomové práce je prakticky ověřit užitečnost rozhodovacích stromů a jejich porovnání s SVM.

V diplomové práci proveďte:
1. Seznamte se s teorií rozhodovacích stromů a teorií SVM. Obojí pak v textové práci také pěkně popište.
2. Zvolte objekt, na němž provedete praktické testování; může se jednat například o postavy.
3. Zvolte příznaky; můžete využít např. histogramu orientovaných gradientů.
4. Implementujte program, pomocí něhož provedete porovnání. Implementaci rozhodovacích stromů, jakož i SVM můžete využít z dostupných knihoven.
5. Sestavte vhodnou trénovací a testovací sadu a proveďte porovnání. Výsledky podrobně zdokumentujte.

Seznam doporučené odborné literatury:

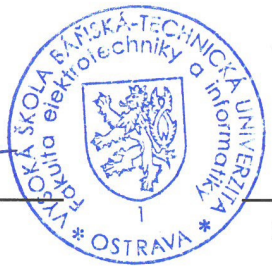
[1] Quinlan, J.R.: Induction of decision trees, Machine Learning 1: 81-106, 1986
Cortes, C., Vapnik V.: Support-Vector Networks. Machine Learning 20: 273–297, 1995
Další dle pokynu vedoucího práce a vlastního průzkumu.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Dr. Ing. Eduard Sojka**

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 28. 6. 2016 ..... Nedošinská

Na tomto místě bych velmi ráda poděkovala svému vedoucímu diplomové práce panu doc. Dr. Ing. Eduardu Sojkovi za možnost vypracování této práce, odborné vedení, ochotu, trpělivost a rady, které mi velmi pomohly k napsání této diplomové práce. Také bych chtěla poděkovat panu Ing. Radovanu Fuskovi, Ph.D. za jeho čas, trpělivost a cenné rady.

Abstrakt

Cílem této diplomové práce je teoretický rozbor dvou známých klasifikátorů, a to rozhodovacích stromů a SVM (Support Vector Machines). Další částí práce je vybrání vhodných příznaků (HOG), jejich praktické použití pro rozhodovací stromy a SVM a jejich následné porovnání. Pro implementaci bylo zvoleno prostředí MS Visual Studio, objektově orientovaný programovací jazyk C++ a knihovna OpenCV.

Klíčová slova: Segmentace obrazu, Gradient, Histogram, Histogram Orientovaných Gradientů, HOG, Rozhodovací strom, Podpůrné vektory, SVM, OpenCV, Detekce, Strojové učení

Abstract

The goal of this diploma thesis is the theoretic analysis of two well-known classifiers, Decision Trees and SVM (Support Vector Machines). Next part of this thesis is choosing the suitable features (HOG), their practical use for Decision Trees and SVM and their comparison. For the implementation we have selected the environment of MS Visual Studio, object oriented programming language C++ and OpenCV library.

Key Words: Image Segmentation, Gradient, Histogram, Histogram of Oriented Gradients, HOG, Decision Tree, Support Vector Machine, SVM, OpenCV, Detection, Machine Learning

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Počítačová grafika je obor, který dokáže nejen upravovat digitální informace získané z reálného světa (např. fotografie), ale také vytvářet zcela nové grafické objekty. Zatímco počítačová grafika dokáže vytvářet objekty a obrazová data pomocí popisů konkrétních objektů, *počítačové vidění* funguje opačným principem, a to tak, že se snaží získávat informace o objektech ze vstupního obrazu.

Počítačové vidění, *zpracování* a *analýza obrazu* mají širokou škálu uplatnění. Jako nejznámější příklad pro širokou veřejnost si můžeme uvést obor medicíny, kde se obrazová data využívají ke stanovení diagnóz pacienta. Z technické stránky můžeme zmínit strojové vidění a podporu výrobního procesu, např. kontrolu kvality a vyhledávání případných defektů výrobků, či detekce objektů pro robota apod. Jako další stojí za zmínku vojenství, kde můžeme zmínit detekci lidí či vozidel, letectví, pro které můžeme zmínit bezpilotní letouny, a nebo také vesmírný výzkum. Jednou z nejdůležitějších částí (nejen) počítačového vidění je *rozpoznávání*, kterému se budeme hlouběji věnovat v této práci. Pro člověka se jedná o snadnou úlohu, kde na základě předchozí zkušenosti dokáže rozpoznat různé objekty ve svém okolí. Z pohledu výpočetní techniky se jedná o náročnější úkol, kde počítač musí nejdříve mít určené specifické *vlastnosti* (*příznaky*) popisující dané objekty. K řešení tohoto problému máme k dispozici tzv. *příznakové metody*, zabývající se popisem, rozpoznáváním a následnou klasifikací objektů na základě příznaků.

Můžeme si také zmínit pojem *umělá inteligence*, která se pokouší o vytváření strojů se známkami inteligentního chování, a jeden z jejích postupů, konkrétně *strojové učení*. Jak lze vytušit z názvu metody, jedná se o algoritmy umožňující počítačům „učit se“. Učením v tomto kontextu myslíme proces, kdy se počítač, kterému dáme nějakou vstupní sadu dat (*trénovací sadu*), snaží „naučit“ předpovídat nová, dříve neviděná data (*testovací sadu*) a zařazovat je do skupin (*klasifikovat data*) na základě podobností jejich atributů. Můžeme si také zavést dva druhy úloh, které budeme později zmiňovat, a to *klasifikace* a *regrese*. Jednoduše řečeno, při klasifikaci se jedná o úlohu, která rozděluje vstupní data do různých tříd (dvou a více), kdežto při regresi odhadujeme číselnou hodnotu výstupu podle hodnoty vstupu.

Strojové učení má velké uplatnění v praxi, nás bude ale nejvíce zajímat jedno z uplatnění, a to detekce objektů v obraze pro účel rozpoznávání obsazenosti parkovacích míst na parkovišti, které má v dnešní době dalekosáhlé využití v praxi, kdy se např. může pomocí aplikace řidič informovat na které parkoviště ve městě má pro něj smysl jet, aneb kde je ještě jedno či více volných a neobsazených míst pro parkování.

V naší práci se zaměříme na dva typy strojového učení, a to *rozhodovací stromy* (*Decision*

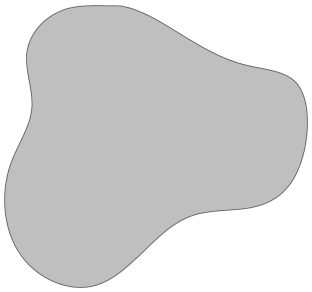
Trees) a *SVM* (*Support Vector Machines*), na kterých pomocí příznaků *HOG* (*Histogramu Orientovaných Gradientů*) otestujeme jejich vhodnost pro detekci obsazenosti parkovacích míst. [1] uvedl rozhodovací stromy, zaměřil se na jednu podoblast učení stroje a na skupinu učících systémů, která byla použita k vytvoření jednoduchého druhu systémů založených na učení. [2] uvedl SVM, jeho otestování a významnost v praxi.

2 Analýza obrazu

Jak jsme nastínili v úvodu, v naší práci se budeme věnovat převážně rozpoznávání objektů ve vstupním obraze. Pro tento účel si tedy nejdříve nadefinujme několik základních pojmů. Jako první uveďme *segmentaci obrazu*, která udává extrahování objektů nacházejících se ve vstupním obraze od pozadí, které nás v danou chvíli nezajímá. V naší práci se bude jednat konkrétně o extrakci aut z parkoviště, tedy jak bylo řečeno - o segmentaci chtěných objektů od pozadí. Při segmentaci jsme se zaměřili na detekci *hran* (nebo také *kontur*), dalším častým postupem ale může být také např. zaměření se na detekci celých oblastí reprezentovaných objektů.

2.1 Detekce hran

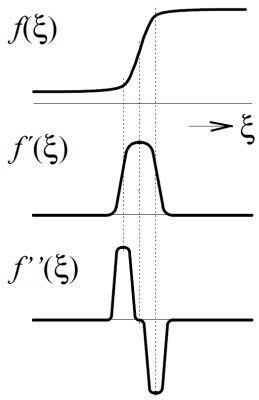
Nyní si více popíšeme postup detekce hran. Zaveďme si pojem *obrazová funkce* $f(x, y)$. Tato funkce, kterou můžeme též nazývat *jasovou funkcí*, nám bude popisovat náš vstupní obraz. Zopakujme si, že jednotlivé objekty jsou ve vstupních obrazech reprezentovány souvislou oblastí (jednoduchou ukázku můžeme vidět na obr. 1) a každá tato oblast musí mít svou *hranici* skládající se z hran neboli křivek. Tyto hrany se skládají z *hranových bodů*, které vytvářejí hranici objektu.



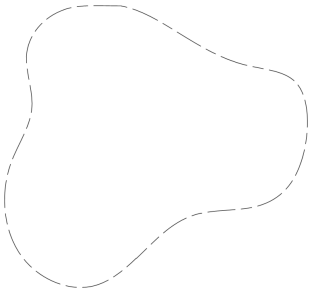
Obrázek 1: Ukázka objektu reprezentovaného oblastí a jeho hranice

Pro úkol nalezení hran a hranic tedy potřebujeme nalézt jednotlivé hranové body. K tomuto účelu se využívá vlastností průběhu jasové funkce, jejíž ukázku můžeme vidět na obr. 2.

V ideálním případě můžeme za bod hrany považovat takový bod, ve kterém má jasová funkce náhlou změnu průběhu, případně bod inflexe. Kdybychom všechny takovéto body nalezli, můžeme je pak spojit do hran a následně do celých hranic. Ideální případ se ovšem od běžně se vyskytujícího případu liší a nastávají různé komplikace. Nejčastěji se v obrazech může vyskytovat např. *šum*, který může způsobit detekování falešných hran, tedy označit za hranové body ty, které hranové být nemají, nebo může např. způsobit vznik neuzavřené hranice objektu (ukázku můžeme vidět na obr. 3).



Obrázek 2: Obrazová (jasová) funkce a její první a druhá derivace ve směru napříč hranou [9]



Obrázek 3: Ukázka nekompletní hranice objektu

2.1.1 Gradientní metody

Vraťme se k obr. 2, na kterém můžeme vidět první a druhou derivaci průběhu jasu ve směru napříč hranou, a v souvislosti s ním si uveďme *gradientní metody*. Jako první se zaměřme na první derivaci průběhu jasu a její hodnotu, která nám určuje velikost nebo také *intenzitu* hrany. V místě hrany má totiž průběh jasu vysokou absolutní hodnotu funkce. Budeme-li chtít popsat velikost hrany, pak si k tomuto účelu zavedeme pojem *hranové operátory*.

Jako nejjednodušší hranové operátory si můžeme uvést derivace ve směru souřadných os x a y, tedy $\partial f/\partial x$ a $\partial f/\partial y$, udávající změnu úrovně jasu ve zmíněných směrech a vhodných pro vyhledávání hran rovnoběžných se zmíněnými osami. Tento postup je sice velmi jednoduchý, ale nepraktický, jelikož často nevíme směr hledané hrany, a tedy ji nemůžeme vyšetřovat pouze ve směrech souřadnicových os. Z tohoto důvodu budeme pro neznámý směr hrany, tzv. *obecný směr*, vyšetřovat průběh jasu nikoli ve směru souřadných os, ale ve směru kolmém na směr potenciální hrany. Zaveďme si vektor $\mathbf{n} = (\cos\theta, \cos\theta)$ popisující tento směr a ξ popisující souřadnici měřenou v tomto směru. Nyní si můžeme určit výpočet derivace ve směru, a to

$$\frac{\partial f}{\partial \xi} \quad = \quad \mathrm{grad}\,(f) \cdot \mathbf{n} = \frac{\partial f}{\partial x} \cdot \cos\theta + \frac{\partial f}{\partial y} \cdot \sin\theta,$$

kde nám $|\partial f/\partial \xi|$ bude udávat velikost hrany.

V běžných případech ovšem většinou směr hrany není předem znám, a tudíž nás pro zkoumaný bod a jeho potenciální hranu bude zajímat směr, ve kterém bude změna jasu nejvyšší, tedy gradient obrazové funkce. Jak jsme již nastínili, výsledný směr hrany bude potom kolmý ke směru gradientu obrazové funkce a velikost hrany nám bude značit velikost tohoto gradientu.

Zaveďme si $e(x,y)$ označující velikost hrany, poté $\varphi(x,y)$ označující směr gradientu a nakonec $\psi(x,y)$ označující směr hrany v bodě (x,y). Z důvodu následného zjednodušení a zpřehlednění vzorců si můžeme zavést substituce za jednotlivé označení derivací, konkrétně $f_x(x,y) = \partial f(x,y)/\partial x$ a $f_y(x,y) = \partial f(x,y)/\partial y$. Po takovémto zjednodušení dostaneme

$$\begin{aligned} e(x,y) &= \sqrt{f_x^2(x,y) + f_y^2(x,y)}, \\ \varphi(x,y) &= \arctan\left[\frac{f_y(x,y)}{f_x(x,y)}\right], \quad \psi(x,y) = \varphi(x,y) + \frac{\pi}{2}. \end{aligned} \tag{1}$$

Zopakujme si, že v praxi se snažíme určit, zda zkoumaný bod daného obrazu leží na hranici objektu či nikoli. K tomu nám může v jednoduchých případech pomoci i určení prahu a jeho porovnání s $e(x,y)$, nicméně hranice objektu mohou být širší než pouze jeden pixel a není zde řešení problému, kdy máme chybějící či nadbytečné hrany, jak jsme zmínili dříve.

Doposud jsme počítali pouze se spojitými funkcemi, v praktických případech se ovšem většinou používají funkce diskrétní

$$\begin{aligned} f_x(x,y) &= f(x+1,y) - f(x,y), \\ f_y(x,y) &= f(x,y+1) - f(x,y), \end{aligned}$$

při kterých je výsledná hodnota $e(x,y)$ porovnávána s prahovou hodnotou, a to většinou pro všechny pixely vstupního obrazu.

Existuje více druhů hranových operátorů - např. Robertsův operátor, operátor Prewittové, Sobelův operátor apod., nicméně ty v naší práci nepoužíváme a proto nám stačí pro naše účely zůstat pouze u gradientů.

Pro detekci hran můžeme využít i druhou derivaci obrazové funkce. Na obr. 2 vidíme, že druhá derivace obrazové funkce ve směru napříč hranou nabývá dvou extrémů, které jsou opačného znaménka, a které se mění v místě hrany. Jak jsme již dříve zmínili, směr hrany nebývá většinou předem znám, a z tohoto důvodu je potřeba počítat druhou derivaci nejméně ve dvou směrech, konkrétně např. ve směru souřadné osy x a poté ve směru souřadné osy y, a to např. za pomoci

Laplaceova operátoru. Znovu můžeme pro zjednodušení a zpřehlednění zápisu zavést označení $f_{xx}(x,y) = \partial^2 f(x,y)/\partial x^2$ a $f_{yy}(x,y) = \partial^2 f(x,y)/\partial y^2$, a poté můžeme Laplaceův operátor zapsat jako

$$\Delta f(x,y) \quad = \quad f_{xx}(x,y) + f_{yy}(x,y).$$

Opět, jedná-li se o diskrétní funkci, nikoli spojitou, pak nahradíme derivace diferencemi a pro body ležící uvnitř obrazu získáme

$$\begin{aligned} f_{xx}(x,y) &= f(x-1,y) - 2f(x,y) + f(x+1,y), \\ f_{yy}(x,y) &= f(x,y-1) - 2f(x,y) + f(x,y+1). \end{aligned}$$

Nyní můžeme Laplaceův operátor vyjádřit jako

$$\Delta f(x,y) \quad = \quad f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y).$$

Pokud bychom chtěli detekovat nikoli hrany, ale celé oblasti, museli bychom si určit nějaké kritérium homogenity oblasti, jako např. velmi podobné hodnoty jasu, barvy apod. Jak jsme již naznačili, metody pro detekci celých oblastí jsou preferovány u obrazů, ve kterých se vyskytuje šum a ve kterých je díky tomu větší šance na chybné určení. Takovéto metody lze rozdělit do tří skupin:

- detekce prahováním,
- detekce narůstáním oblastí a
- detekce dělením oblastí.

Jelikož se v této práci zaměříme na detekci hran, nikoli oblastí, není potřeba detekci oblastí popisovat detailněji.

2.2 Histogram jasu

Zopakujme si, že $f(x,y)$ značí jasovou funkci pro vstupní obraz a označme $g(x,y)$ jako funkci pro obraz výstupní. Zaveďme si pojem *bodová operace*, která značí nějakou transformaci vstupního obrazu, což můžeme zapsat jako

$$g(x,y) \quad = \quad \varphi[f(x,y)].$$

Pomocí bodových operací tedy získáme výstupní obraz takovým postupem, že pro každý bod o souřadnicích (x,y) přiřadíme takovou hodnotu jasu, která je pro bod o stejných souřadnicích

(x, y) nějakou funkcí φ pro jas ze vstupního obrazu.

Nyní si zaveďme důležitý pojem *histogram jasu*. Mějme diskrétní obraz obsahující pouze celočíselné hodnoty, definovaný na oblasti z N bodů. Má-li jas hodnotu z, pak počet bodů obsahujících právě tuto hodnotu je N_z, a poté pro histogram jasu H platí $H(z) = N_z$. Označíme-li $p(z)$ jako pravděpodobnost jevu, kdy hodnota jasu daného bodu bude právě z, pak můžeme tvrdit, že $p(z) = H(z)/N = N_z/N$. Hodnotu jasu v daném bodě můžeme označit jako náhodnou proměnnou $\mathbf{b}$. Tuto diskrétní úlohu je také možno převést na úlohu na spojitých obrazech, které obsahují reálné hodnoty. Pravděpodobnost jevu, že jas v konkrétním bodu nabývá hodnoty z intervalu $(z, z + \Delta z\rangle$, označujeme jako $\mathscr{P}\{z < \mathbf{b} \leq z + \Delta z\}$. Plochu celého obrazu můžeme označit jako $\mathscr{S}$ a $\mathscr{S}\{z < \mathbf{b} \leq z + \Delta z\}$ můžeme označit jako konkrétní plochu pro nějakou část z celého obrazu, pro kterou platí, že její jas nabývá hodnoty z intervalu $(z, z + \Delta z\rangle$. Pro spojitý případ tedy $p(z)$ značí hustotu pravděpodobnosti

$$p(z) = \lim_{\Delta x \to 0} \frac{\mathscr{P}\{z < \mathbf{b} \leq z + \Delta z\}}{\Delta z} = \lim_{\Delta x \to 0} \frac{\mathscr{S}\{z < \mathbf{b} \leq z + \Delta z\}}{\mathscr{S}\Delta z}.$$

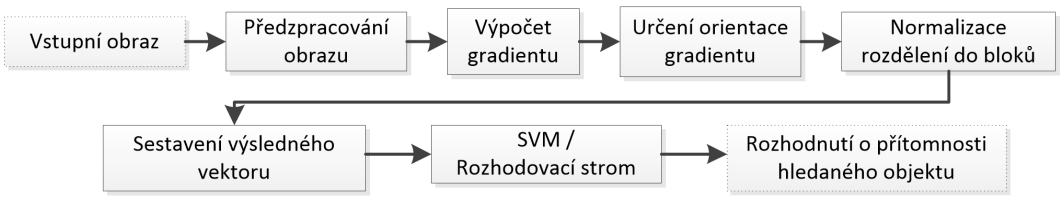
Jedna z možných úprav histogramu jasu je jeho *vyrovnání*, které se použije v případě, kdy chceme získat rovnoměrné rozložení hustoty pravděpodobnosti jasu po celém histogramu pro výsledný obraz. Důvody, proč tuto akci provádíme s histogramy jasu, mohou být různé, většinou je to však pro zajištění normalizované jednotné podmínky např. před segmentací a rozpoznáváním objektů v obraze. Ve většině případů vyrovnání histogramu vede k jakémusi zlepšení kvality obrazu, nicméně toto zlepšení neplatí vždy, pro některé případy se může stát, že vyrovnáním histogramu tuto kvalitu zhoršíme.

2.3 Histogram orientovaných gradientů

Metoda *histogramu orientovaných gradientů* (*HOG*, *Histogram of oriented gradients*), představena v [3], používá pro popis objektů příznakové vektory. Jak jsme nastínili v úvodu práce, příznaky slouží k zaznamenání důležitých vlastností (informací), popisují vstupní obraz a objekty v něm a naším účelem je nalézt takové, které co nejlépe popisují hledaný objekt (v našem případě automobil na parkovacím místě) ve vstupních obrazech. K popisu obrazu nám pomohou vztahy (1). Objekt může být charakterizován podle svého rozdělení intenzity gradientů nebo směrů hran, a to dokonce i bez přesné znalosti odpovídajícího gradientu nebo pozic hran. Pomocí získaných hodnot sestavujeme histogramy k určení hledaného objektu v obraze.

Na obr. 4 můžeme vidět algoritmus metody HOG. Krok předzpracování obrazu může být různý, záleží na požadavcích a vhodnosti aplikovaných metod. My jsme použili převod vstupního obrazu z RGB do stupňů šedi, normalizaci, rozostření a také změnu velikosti pro náročné počítání programu, jelikož máme velkou sadu vstupních obrazů jak pro trénování, tak pro testování,

tudíž pro zefektivnění a urychlení výpočtu jsou tyto úpravy zapotřebí.



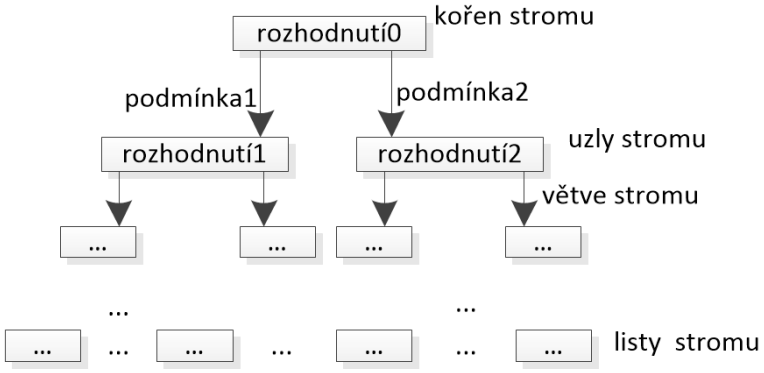
Obrázek 4: Algoritmus metody Histogram orientovaných gradientů (HOG)

Máme-li vstupní obraz předzpracován, můžeme jej rozdělit do malých prostorových oblastí, kterým říkáme *buňky* (síť stejně velkých čtvercových bloků), a pro každou tuto oblast (všechny pixely v buňce) vypočítáme jednorozměrný histogram směrů gradientů. Tyto gradienty pak určují daný objekt ve vstupním obraze. Informace můžeme shromažďovat z větších částí, nazývající se *bloky*, které obsahují několik buněk z obrazu. Bloky se mohou navzájem překrývat, což může vést ke zlepšení detekce, nicméně i k větší časové náročnosti algoritmu.

HOG má velké úspěchy pro detekci osob, pro které má i v OpenCV naimplementované metody, my jsme si HOG vybrali pro implementaci jak SVM, tak i pro rozhodovací stromy a otestovali jeho funkčnost pro oba klasifikátory na obrazech obsahující parkovací místa místo lidí. V kapitole 6 byla navrhnuta sada testů pro otestování rozhodovacích stromů i SVM v souvislosti s HOG a zhodnoceny úspěšnosti jednotlivých metod.

3 Rozhodovací stromy

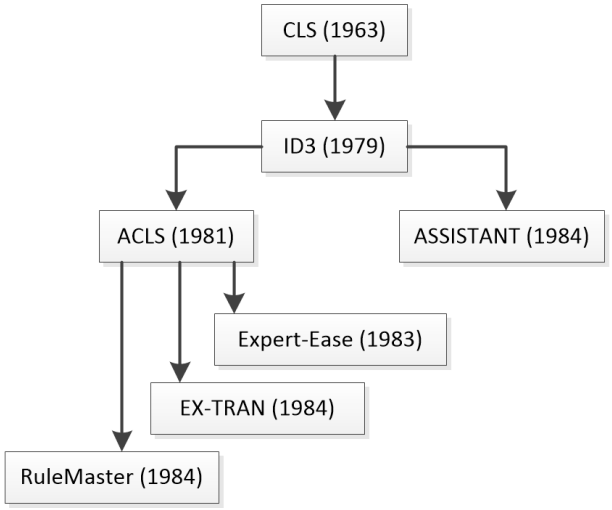
Rozhodovací strom (Decision Tree) je *orientovaný graf*, ve kterém jeho výchozí horní uzel nazýváme *kořenem* stromu a každá jeho větev končí *listem* stromu (nebo také *terminálním uzlem*), tedy uzlem, který se již dál nevětví. Každý z jeho vnitřních uzlů je reprezentován podmínkou a každý listový uzel reprezentuje rozhodnutí, jedná se tedy o sadu hierarchicky uspořádaných rozhodovacích pravidel, kde se rozhodovací strom větví, nicméně větve se pak již znovu nespojují. Kořen stromu obsahuje první otázku nebo také rozhodnutí, pomocí kterého procházíme stromovou strukturou dále po jednotlivých hranách na základě našich rozhodnutí až dojdeme k listu stromu, který nám určuje konečnou odpověď na původní otázku. Ukázku struktury rozhodovacího stromu můžeme vidět na obr. 5. Jako další důležitý pojem si můžeme zmínit *výšku* stromu neboli také jeho *hloubku*, označující počet úrovní stromu od kořene stromu k nejvzdálenějšímu listu stromu.



Obrázek 5: Struktura rozhodovacího stromu

3.1 Počáteční vývoj

Jako počáteční vývoj si můžeme označit skupinu TDIDT („Top-Down Introduction of Decision Trees“), kterou můžeme vidět na obr. 6 a krátce si představíme jednotlivé její postupy. Jako první byl Concept Learning System framework (CLS) [4], který vytváří rozhodovací strom snažící se minimalizovat cenu klasifikace objektu. Tato cena má složky dvou typů, první je měření ceny určující hodnotu vlastnosti A, kterou se projevuje objekt, a druhou je špatná klasifikace ceny rozhodující, zda objekt patří do třídy J i přesto, že jeho správná třída je K. CLS používá plánovací strategii podobnou minimu z maxima. V každé fázi CLS prozkoumává prostor možných rozhodovacích stromů pro fixní hloubku, vybírá akci k minimalizaci ceny v tomto omezeném prostoru, a poté se posune o jednu úroveň níže ve stromě. V závislosti na hloubce stromu může CLS vyžadovat značné množství výpočtů, nicméně je schopno objevit nepatrné vzory, které mu byly dány.



Obrázek 6: TDIDT [1]

ID3 [5] a [6] je jeden z programů vyvinutých z CLS jako odpověď na náročný indukční úkol představen Donaldem Michie a má schopnost rozhodnout se ze vzorově založených příznaků.

ACLS [7] je zobecnění ID3. CLS a ID3 požadují nárok na to, aby každá vlastnost, která byla použita pro popsání projektů, měla pouze hodnoty z dané sady. Kromě vlastností tohoto typu pak ale ACLS umožňuje vlastnostem mít neomezené číselné hodnoty, což umožnilo této metodě pracovat např. na rozpoznání obrazu.

ASSISTANT [8] dále zobecňuje celočíselné hodnoty ACLS, a to díky umožnění práce s atributy se spojitými (reálnými) hodnotami. Tato metoda umožňuje třídám vytvoření hierarchie, kde jedna třída může být jemnějším rozdělením druhé třídy a nevytváří rozhodovací strom iterativně ve způsobu použití ID3, nicméně zahrnuje algoritmy pro volbu „dobré“ trénovací sady z dostupných objektů.

Poslední tři algoritmy na obr. 6 jsou komerčně odvozené od ACLS. Neobsahují významně pokročilejší základní teorii, nicméně obsahují několik vylepšení, které urychlují vytváření a používání rozhodovacích stromů.

3.2 Teorie rozhodovacích stromů

Nyní si podrobněji popíšeme teorii rozhodovacích stromů. Mějme svět objektů, ve kterém jsou tyto objekty popsány pomocí pojmů souborů atributů a každý takovýto atribut označuje nějakou důležitou vlastnost daného objektu. Řekněme, že si chceme do stávající smečky pořídit nového potkana. Budeme mít tedy objekt „nový potkan“ na který budeme aplikovat klasifikační úlohu na základě jeho vlastností. Příklad atributů takového objektu může např. být

- typ srsti s hodnotami standard, velveteen a rex,
- barva srsti s hodnotami bílá, šedá a černá,
- uši s hodnotami standard a dumbo a
- oči s hodnotami černé a červené.

Tyto atributy společně tvoří jazyk nultého řádu a slouží k charakterizaci objektů v daném světě. Nějaký konkrétní objekt „nový potkan“ by tedy mohl vypadat následovně

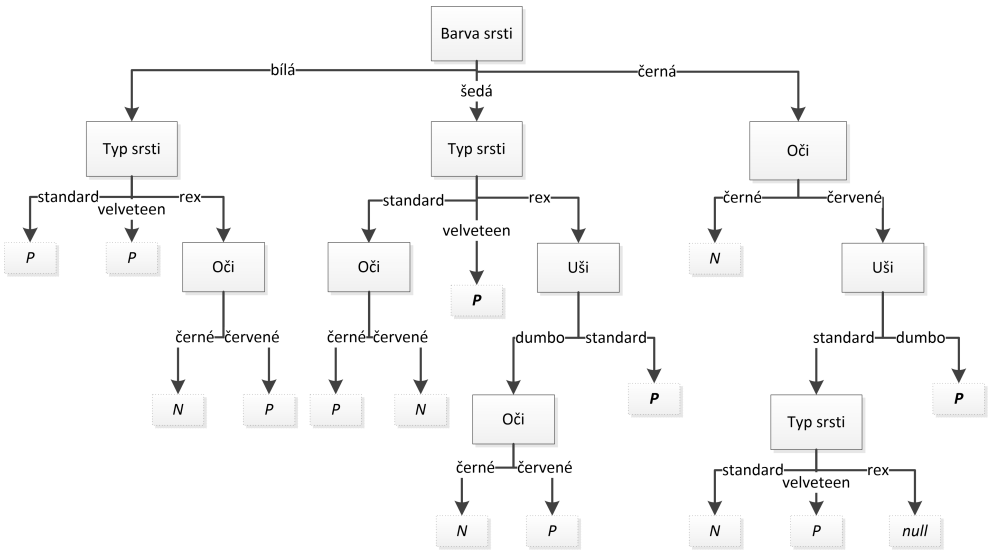
- typ srsti: velveteen,
- barva srsti: šedá,
- uši: dumbo,
- oči: červené.

Každý objekt v daném světě pak můžeme zařadit do nějaké třídy, která jej charakterizuje. Pro zjednodušení můžeme dále počítat pouze se dvěma třídami P a N. Cílem algoritmu je vytvořit takové klasifikační pravidlo, které bude schopno pro jakýkoli objekt určit jeho třídu a to za pomocí zmíněných hodnot atributů. K tomuto procesu je důležitá trénovací sada objektů, kterou jsme si definovali v úvodní části práce a pro kterou jsou známé třídy jednotlivých objektů. Otázkou zůstává, zda atributy, které si nadefinujeme, jsou dostačující co se týče informačního zisku pro klasifikaci objektu do dané třídy. Pro případ si můžeme uvést trénovací sadu, ve které jsou objekty (dva a více), které mají stejné hodnoty jednotlivých atributů, nicméně každý z nich patří do jiné třídy. V takovémto případě je téměř nemožné takovéto objekty správně klasifikovat a rozlišovat mezi nimi jen za pomoci hodnot atributů, hodnoty atributů budou tedy označeny za nedostatečné pro trénovací sadu.

Rozhodovací strom tedy vyjadřuje nějaké klasifikační pravidlo pro daný svět objektů. V tab. 1 můžeme vidět ukázku malé trénovací sady objektů, resp. různých hodnot atributů pro objekt „nový potkan“ a rozhodovací strom k této tabulce můžeme vidět na obr. 7. Při klasifikaci se vždy začíná u kořenového uzlu stromu a po vyhodnocení testu (otázky) pokračujeme ve stromové struktuře tou větví, která odpovídá výsledku. Pro binární stromy, kterým se budeme věnovat v praktické části, je to vždy jedna ze dvou možností. Můžeme tedy ve stromové struktuře pokračovat směrem vlevo anebo vpravo, podle toho, jaké otázky jsme si v jednotlivých uzlech nadefinovali. Takto pokračujeme postupem přes jednotlivé větve stromu až k listovému uzlu, který nám určí hledanou třídu objektu. Příklad objektu „nový potkan“, který jsme si uvedli na začátku, není součástí trénovací sady a měl by patřit do třídy P.

Číslo atributu	Typ srsti	Barva srsti	Uši	Oči	Třída atributu
1	standard	černá	standard	červené	N
2	standard	černá	standard	černé	N
3	velveteen	černá	standard	červené	P
4	rex	šedá	standard	červené	P
5	rex	bílá	dumbo	červené	P
6	rex	bílá	dumbo	černé	N
7	velveteen	bílá	dumbo	černé	P
8	standard	šedá	standard	červené	N
9	standard	bílá	dumbo	červené	P
10	rex	šedá	dumbo	červené	P
11	standard	šedá	dumbo	černé	P
12	velveteen	šedá	standard	černé	P
13	velveteen	černá	dumbo	červené	P
14	rex	šedá	dumbo	černé	N

Tabulka 1: Ukázka malé trénovací sady



Obrázek 7: Rozhodovací strom pro objekty z tab. 1

Jestliže budeme mít adekvátní atributy objektů, pak bude vždy možné sestrojit správně klasifikující rozhodovací strom pro všechny objekty z trénovací sady. Jak můžeme předpokládat, většinou neexistuje pouze jeden správně klasifikující strom, ale lze vytvořit více různých rozhodovacích stromů, které budou všechny správně klasifikovat. Nyní máme tedy stromy, které správně klasifikují trénovací sadu objektů a naším cílem je se posunout dále, tedy na takové rozhodovací stromy, které budou správně klasifikovat nejen trénovací sadu, ale také objekty nové, tedy testovací sadu objektů. Z tohoto důvodu je důležité zachytit nějaký smysluplný vztah mezi třídou objektu a jejími hodnotami atributů.

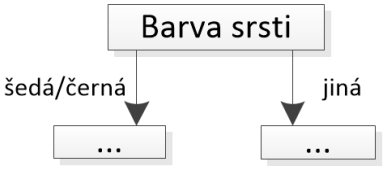
Máme-li ale obrazy získané z reálného světa, často se nám stane, že jsou poškozeny šumem. Jedná se o chyby v atributech nebo třídách objektů zapřičiňující špatnou klasifikaci daného objektu. Při práci se sadou objektů poškozenou šumem máme dvě podmínky pro algoritmus, aby s ní byl schopen dále pracovat. První je schopnost práce s nedostatečnými atributy, tedy atributy poškozenými šumem a vypadajícími nevhodně ke klasifikaci. Druhá podmínka je schopnost algoritmu se rozhodnout, kdy již testování dalších atributů nemůže vylepšit prediktivní přesnost rozhodovacího stromu. Obecně se snažíme o to, aby se nám nezvyšovala složitost rozhodovacího stromu kvůli případům se šumem. S těmito problémy nám může pomoci porovnávání s nějakým určeným prahem, tedy jakési odfiltrování irelevantních atributů, nicméně s možným rizikem odfiltrování i relevantních atributů.

Pro vytváření stromů existuje v dnešní době řada různých algoritmů. My jsme se v naší praktické práci zaměřili na binární rozhodovací stromy, a proto si v teoretické části více popíšeme algoritmus CART. Obecně je princip vytváření různých rozhodovacích stromů podobný, lišící se např. ve vhodnosti nalezeného prediktoru apod.

3.3 CART

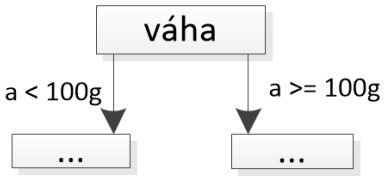
CART (*Classification and Regression Trees*) jsou binární rozhodovací stromy. Pojďme si nadefinovat několik důležitých pojmů před hlubším ponořením se do teorie těchto rozhodovacích stromů. Proměnnou T označíme náš rozhodovací strom, který bude obsahovat sadu uzlů $t = (t_1, \ldots, t_N)$. Dále si zaveďme pojem závisle proměnné Y a pro klasifikační rozhodovací stromy kategorie $c = (c_1, \ldots, c_C)$, kde $C \geq 2$, jelikož se jedná o binární dělení a nejméně kořen stromu nabízí tedy rozdělení vstupní sady objektů do dvou kategorií. Pro spojitou závisle proměnnou $Y = (y_1, \ldots, y_n)$ máme regresní rozhodovací strom a hodnota, která bude přiřazována, je $\widehat{y}_i$. Jednotlivá pozorování závisle proměnné Y jsou rozdělena do uzlů rozhodovacího stromu pomocí hodnot vysvětlujících proměnných, nazývaných též prediktory, $X_1, \ldots, X_M$. Pro kategoriální prediktory, jako v příkladě uvedeném v teoretické části této práce, jsou pro spojitou závisle proměnnou $Y = (y_1, \ldots, y_n)$ jednotlivé hodnoty y_i rozděleny podle kategorií prediktoru X. Na obr. 8 můžeme vidět ukázku velmi jednoduchého rozhodovacího stromu z našeho známého příkladu objektu „nový potkan“, ve kterém máme větvení s prediktorem „Barva srsti“ a dvě možné kategorie, do první řadíme ty potkany, které mají buď šedou nebo černou srst, do druhé s jakoukoli jinou barvou. Vytvoříme tedy z kořenového uzlu z proměnné Y dva dceřiné uzly, které se pak mohou dále binárně větvit na základě dalších definovaných podmínek. Jedná se tedy o takovou skupinu stromů, ve které nás zajímá, zda pozorování y_i patří do množiny, ve které $x_i \in A$, kde $A \neq \emptyset$ a je vlastní podmnožinou množiny všech hodnot X.

Nyní si ukažme jednoduchý rozhodovací strom pro spojitou proměnnou, který můžeme vidět na obr. 9. Závisle proměnnou Y zde rozdělujeme pomocí hodnoty a daného prediktoru X. Ukázka



Obrázek 8: Ukázka jednoduchého kategoriálního rozhodovacího stromu

nám znázorňuje rozhodování se na základě váhy objektu „nový potkan“, kde si jako hranici určíme 100 g. Bude-li mít potkan nižší váhu, může se např. jednat o mládě, jinak by se mohlo jednat o dospělého jedince. Samozřejmě se tento rozhodovací blok dá navázat před či za další rozhodovací faktory jako pohlaví apod., nicméně pro teoretické účely nám stačí tato malá ukázka.



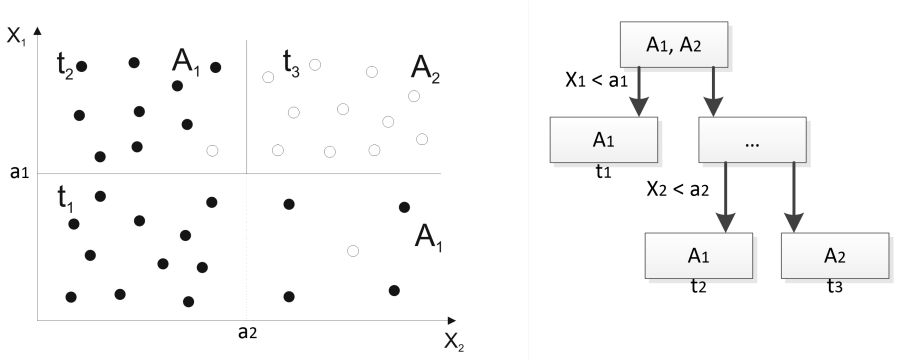
Obrázek 9: Ukázka jednoduchého spojitého rozhodovacího stromu

Při každém větvení na daném uzlu se rozhodujeme na základě jednoho konkrétního prediktoru X, což nicméně nevylučuje možnost použití stejného prediktoru i v dalších rozhodováních, např. kdybychom chtěli určovat ještě jemněji rozmezí vah potkana. Musíme ovšem počítat s tím, že nám pak narůstá velikost celého rozhodovacího stromu a volit tedy moudře jednotlivá rozhodování. V každém jednotlivém listovém uzlu stromu máme uloženo pozorování y_i. Pro klasifikační rozhodovací stromy přiřazujeme do listového uzlu kategorii a pro případ regresních rozhodovacích stromů zde přiřazujeme průměr hodnot závisle proměnné Y.

Jelikož u rozhodovacích stromů není nátlak na rozložení vstupních dat, jsou parametry rozhodovacího stromu v naší praktické části určovány experimentálně na základě testování různých hodnot a vlastností rozhodovacího stromu.

Nyní, když jsme si podrobně nadefinovali jednotlivé pojmy pro následující práci s rozhodovacími stromy, si můžeme přiblížit jak fungují stromy typu CART. Při začátku algoritmu typu máme vždy celou sadu (trénovací sadu) k dispozici a ta vstupuje do kořenového uzlu rozhodovacího stromu. Každý nelistový uzel (včetně toho kořenového) se poté dělí na právě dva dceřiné uzly a to vždy podle nějaké hodnoty a prediktoru X. Tento proces nazýváme růst rozhodovacího stromu. Při větvení nám hodnoty vysvětlujících proměnných budou rozdělovat náš prostor na podprostory (můžeme si je představit jako pravoúhelníky, viz obr. 10), a to nejméně dva po průchodu přes kořenový uzel, který prostor rozdělí na dva podprostory a případně pak v

jednotlivých podprostorech pokračujeme dělením na další v závislosti na hloubce rozhodovacího stromu.



Obrázek 10: Ukázka rozhodovacího stromu a jeho dělení prostoru na podprostory

V následující části si představíme tzv. *kriteriální statistiku* (*spliting criterium*), která nám určuje homogenitu daného uzlu. Tato část je důležitá pro zajištění co nejlepšího rozdělení pomocí prediktoru a jeho hodnoty.

3.3.1 Kriteriální statistika

Nyní si ukážeme nejčastější měřené kriteriální statistiky pro regresní strom. Tento strom budeme mít rozdělen pomocí terminálních uzlů o určitém počtu a jako konstantu vyjádříme predikovanou hodnotu závisle proměnné Y, a to pro každý jednotlivý dceřiný uzel. Kritérium, které použijeme, bude minimalizovat střední kvadratickou chybu a nejlepším odhadem této konstanty bude průměr. Naším cílem je najít takové rozdělení závisle proměnné Y, které nejmenší průměrnou kvadratickou odchylku hodnot y_i bude mít co nejmenší v potenciálním uzlu t od průměru hodnot y_i.

Výpočet kritéria minima kvadratické chyby $Q\,(T)$ je tedy

$$\begin{aligned} \overline{y_t} &= \frac{1}{N_t}\sum y_{i(t)} \\ Q_t\,(T) &= \frac{1}{N_t}\sum_{i=1}^{N_t}\left(y_i - \overline{y_t}\right)^2, \end{aligned}$$

kde N_t značí počet pozorování v uzlu t a $y_{i(t)}$ označují hodnoty závisle proměnné Y v uzlu t.

Druhá kriteriální statistika, kterou si ukážeme, se týká klasifikačních stromů. Její princip je založen na poměru kategorií závisle proměnné v jednotlivých potenciálních uzlech. Mezi nejčastěji používanými kritérii je Gini index (GI), Entropie (H) a klasifikační chyba (ME)

a dají se vypočítat následovně

$$\begin{aligned} \text{Gini index} : GI &= \sum_{c=1}^{J} p_{tc}\,(1 - p_{tc}) = 1 - \sum_{c=1}^{J} p_{tc}^2, \\ \text{Entropie} : H &= -\sum_{c=1}^{J} p_{tc} \log_2 p_{tc}, \\ \text{Klasifikacni chyba} : ME &= 1 - \max\{p_{tc}\}, \end{aligned}$$

kde p_{tc} značí podíl pozorování y_i s kategorií c v uzlu t z celkového počtu veškerých pozorování y_i v určitém uzlu. Jedná se tedy o pravděpodobnost kategorie c v uzlu t.

Použití Gini indexu (GI) se vyskytuje nejčastěji jako kriteriální statistika pro klasifikační stromy typu CART. Jeho výsledná hodnota bude nula, jestliže se bude v konečném uzlu vyskytovat pouze jediná kategorie proměnné Y. Naopak jeho hodnota bude maximální, jestliže v konečném uzlu bude v každé jednotlivé kategorii nezávisle proměnné Y stejný počet pozorování.

Při každém rozdělování uzlu na dceřiné uzly musíme vypočítat GI pro každý takový dceřiný uzel. Pro výslednou hodnotu Gini indexu GI_{vys} vypočítáme vážený součet všech GI všech dceřiných uzlů podle jejich velikostí. Jedná se tedy o součet $GI(i)$ všech dceřiných uzlů, který vynásobíme odpovídajícím podílem pozorování v konkrétním dceřiném uzlu ze všech pozorování v původním rodičovském uzlu. GI_{vys} se tedy vypočítá jako

$$GI_{\text{vys}} = \sum_{i=1}^{k} \frac{N_i}{N_t} GI(i),$$

kde k značí celkový počet dceřiných uzlů (tedy $k = 2$ pro naše binární stromy), N_t značí počet pozorování v rodičovském uzlu t a N_i značí jednotlivé počty v dceřiných uzlech.

Jako další máme Entropii, která nabývá maximální hodnoty ve chvíli, kdy jsou jednotlivé kategorie závisle proměnné Y zastoupeny rovnoměrně v jednotlivých uzlech stromu. Minimální hodnoty Entropie nabývá ve chvíli, kdy v uzlu stromu máme pouze pozorování z jedné kategorie. Jednotlivé Entropie se počítají pro každý dceřiný uzel a výslednou Entropii H_{vys} pro konkrétní dělení pak vypočítáme pomocí váženého součtu jednotlivých Entropií $H(i)$ ve všech dceřiných uzlech

$$H_{\text{vys}} = \sum_{i=1}^{k} \frac{N_i}{N_t} H(i),$$

Časté použití kritéria Entropie je u algoritmu $C4.5$.

Pokles v Entropii můžeme změřit pomocí kritéria informačního zisku $GAIN$ (information gain).

$$\text{zisk}_{\text{vys}} \quad = \quad H - \left(\sum_{i=1}^{k} \frac{N_i}{N_t} H(i) \right)$$

U kritéria Klasifikační chyby (ME) se jedná o podíl chybně klasifikovaných pozorování, celková přesnost stromu pak je $1 - ME$ (takto získáme podíl správně klasifikovaných pozorování). ME se často používá pro finální měření přesností klasifikačních stromů. ME_{vys} (výsledná klasifikační chyba) se vypočítá jako vážený součet ME pro dané dělení v jednotlivých dceřiných uzlech stromu:

$$ME_{\text{vys}} \quad = \quad \sum_{i=1}^{k} \frac{N_i}{N_t} ME(i).$$

3.3.2 Přiřazení hodnoty terminálním uzlům

Pro klasifikační stromy se výsledná kategorie určuje podle toho, která kategorie má největší zastoupení v daném uzlu, tudíž klasifikaci nového objektu provedeme díky kategorie uzlu, do kterého je nový objekt zařazen v průběhu algoritmu rozhodovacího stromu.

Pro regresní stromy je v listovém uzlu stromu přiřazen průměr hodnot závisle proměnné a jejich variabilita, díky které můžeme určit, jak moc přesná je predikovaná hodnota. Nevýhodou pro regresní stromy je výsledná plocha, která je nespojitá. Jedním z možných řešení tohoto problému odstranění nespojitosti je např. proložení dat lineárním regresním modelem v každém jednotlivém terminálním uzlu. Nevýhoda tohoto postupu u regresních stromů je jeho časová náročnost. Proložení dat lineárním regresním modelem také nemusí být vždy uskutečnitelné. Jedná se o takové případy, kdy terminální uzly neobsahují dostatečný počet vzorků nutných pro regresi nebo není mezi nimi nalezen žádný vztah. Zmíněnou nevýhodu regresních stromů dokáží vyřešit jiné stromové metody. Mezi takové patří např. regresní lesy nebo MARS metoda.

3.3.3 Růst stromu

Nyní si ukážeme průběh růstu stromu typu CART v jednotlivých krocích:

1. Rozdělení souboru na dvě části - na trénovací a testovací soubor.

2. Nalezení nejlepšího rozdělení každého z prediktorů:

   (a) Seřazení hodnot každého prediktoru pro spojité proměnné od nejmenších hodnot po největší hodnoty. Po průchodu všemi hodnotami prediktoru X následuje spočítání kriteriální statistiky všech možných různých rozdělení pro proměnnou Y pro potenciální dceřiné uzly. Rozhodování, do jakého uzlu patří pozorování y_i, probíhá pomocí

rozhodnutí, zda je dělicí hodnota a pro prediktor Y menší nebo rovna (případně větší nebo rovna) hodnotě x_i. Pokud ano, může se pozorování uložit do pravého (případně levého) uzlu stromu. Nejmenší hodnota a kriteriální statistiky se vybere pro nejlepší dělení závisle proměnné Y pomocí daného prediktoru. Hodnota a je poté použita pro rozdělení souboru, tedy jednotlivých hodnot y_i, do dvou dceřiných uzlů.

(b) Pro nalezení nejlepšího rozdělení pro kategoriální prediktor se musí projít všechny možné kombinace jednotlivých kategorií prediktoru a hodnot nebo jednotlivých kategorií závisle proměnné. Pro dělení se pak využije nejnižší hodnota kriteriální statistiky.

3. Podle hodnoty v předešlém kroku (2) se soubor rozdělí na dva dceřiné uzly t_1 a t_2.

4. Kroky 2 a 3 se opakují do doby, než se dělení zastaví na hodnotě, která je předem definována. Jedná se o některé z pravidel pro zastavení růstu stromu, které si nadefinujeme v následující podkapitole.

5. Posledním krokem je otestování, neboli použití testovacího souboru. To se provádí za účelem ověření vhodné velikosti stromu. Pokud nastane situace, že strom je příliš velký, pak se musí prořezat.

3.3.4 Pravidla pro zastavení růstu stromu

Pravidla pro zastavení růstu stromu (*stopping rules*) jsou velmi důležitá, jelikož stromy nemohou růst donekonečna. Maximální velikost stromu je dána velikostí souboru.

Zastavení růstu stromu bez vnějšího omezení může nastat v případě, že terminální uzel obsahuje jen jedno pozorování, v případě, že všechna jednotlivá pozorování v uzlu stromu obsahují stejnou hodnotu všech prediktorů, anebo v případě, že všechna jednotlivá pozorování v uzlu stromu obsahují stejnou hodnotu závisle proměnné.

Zastavení růstu stromu pomocí nastavení určitých omezujících parametrů může být např. omezením počtu větvení stromu, omezením počtu pozorování v koncovém uzlu, nebo omezením velikosti chyby v jednotlivých potenciálních dceřiných uzlech stromu.

Použijeme-li nějaké pravidlo pro zastavení růstu stromu, pak náš strom bude mít velikost závislou na tomto pravidle. Výsledek tedy může díky tomu být subjektivní. Naší snahou bude výběr stromu, který bude optimální.

3.3.5 Výběr optimálního stromu

Problém subjektivnosti stromu se řeší rozdělením souboru na trénovací a testovací. Pomocí trénovacího souboru se strom učí a také pomocí něj roste, kdežto pomocí testovacího souboru strom

pouze testujeme.

Existují dvě nechtěné možnosti, které mohou u stromů nastat, a to strom, který je nedoučený (*underfitting*) a strom, který je přetrénovaný (*overfitting*). Nedoučený strom poznáme tak, že je až příliš jednoduchý. Také chyba takovéhoto stromu na obou souborech (testovacím i trénovacím) bude příliš velká. Přetrénovaný strom je naopak příliš složitý, a testovací chyba bude příliš velká, kdežto trénovací naopak zase malá.

Řešením tedy je nalezení vhodného kompromisu mezi těmito dvěma extrémy. Nesmíme také zapomínat na složitost stromu, kterou je nutno vzít v úvahu. Zde se používá obecné pravidlo, tzv. Occamova břitva, která nám určí pravidlo, že máme-li podobnou chybu u dvou modelů, pak je vhodnější vybrat ten model, který je méně složitý.

3.3.6 Prořezávání stromu

Velikost stromu souvisí s jeho složitostí. Máme-li příliš malý strom, může se stát, že nedokáže obsáhnout veškerou informaci v datech. Naopak příliš velký strom může ztrácet svou obecnou platnost.
Jak jsme již zmínili, naším cílem je najít strom optimální velikosti. Toho můžeme dosáhnout takovým způsobem, že nejprve necháme růst strom pomocí nějakého pravidla, např. pomocí určení určité hranice počtu pozorování v uzlu a takovýto velký strom poté prořežeme. Určit optimální strom lze více postupy, jedním z nich může být kritérium složitosti stromu (*cost-complexity criterium*).

Mějme zadán strom T_0 a strom T_1, který získáme prořezáním určitého počtu koncových uzlů ve stromu T_0. Poté je kritérium složitosti stromu:

$$C_\alpha(T_1) \quad = \quad DT_1 + \alpha |T_1|, \tag{2}$$

kde $|T_1|$ značí počet terminálních uzlů stromu T_1 a chybu stromu T_1 značí DT_1. Parametr α, $\alpha \geq 0$, určuje kompromis mezi přesností stromu a velikostí stromu. Ke každému parametru α poté hledáme strom $T_\alpha \subseteq T_0$, který bude minimalizovat $C_\alpha(T)$. Ke zjištění odhadu α se využívá křížová validace (více v [10]).

3.3.7 Přesnost stromu

Mějme $e(t)$ značící chybu na trénovacím souboru, pak $e'(t)$ značí chybu na souboru testovacím. Použijeme-li pouze trénovací soubor, pak můžeme získat dva odhady na celkovou chybu stromu. Tyto dva odhady můžeme rozdělit na pesimistický a optimistický odhad. Pro pesimistický odhad

předpokládáme pro každý terminální uzel chybu na testovacím souboru jako $e'(t) = (e(t) + 0,5)$. Pro optimistický odhad předpokládáme rovnost souborů, tedy $e'(t) = e(t)$, kde se chyba na testovacím souboru rovná chybě na souboru trénovacím. Z tohoto můžeme určit celkovou chybu stromu, která se vypočítá jako

$$e'(T) = e(T) + N \times 0,5, \tag{3}$$

kde N značí počet terminálních uzlů.

Použití pouze trénovacího souboru pro odhad celkové chyby stromu není ovšem nejvhodnějším postupem. Pro odhad celkové chyby stromu je vhodnější použít testovací soubor. Pomocí trénovacího souboru totiž nemusíme vhodně odhadnout, jak moc dobře bude strom klasifikovat nová data.

3.3.8 Přesnost rozhodovacího stromu

Pro klasifikační stromy si jako příklad metody pro určení přesnosti uvedeme Celkovou správnost OA (*overall accuracy*), pomocí které můžeme vypočítat pravděpodobnost, že pozorování bylo klasifikováno správně. Ta se vypočítá jako poměr počtu správně klasifikovaných pozorování n_p a celkového počtu pozorování n.

$$OA = \frac{n_p}{n} \tag{4}$$

Při takovémto měření ovšem může docházet k podhodnocení nebo nadhodnocení kvality modelu, jelikož měření nebere v úvahu rozdílnost oproti náhodnému výsledku a různé velikosti skupin. Můžeme zajistit úpravu velikosti kategorií následovně:

$$OA_{\text{kateg}} = \frac{1}{J}\sum_{c=1}^{J}\frac{n_{pc}}{n_c}, \tag{5}$$

kde J značí celkový počet kategorií, n_{pc} značí v kategorii c počet správně klasifikovaných pozorování a n_c značí v kategorii c počet všech pozorování.

Celková správnost OA je vhodná pro srovnávání s ostatními klasifikačními metodami. Další použití může být pro výběr vhodného stromu, nejčastěji se ale používá pro určení procenta správně klasifikovaných pozorování a to pro každou jednotlivou kategorii.
Optimistický odhad chyby na trénovacím souboru pak můžeme značit jako $e'(t) = 1 - OA_{\text{tren}}$.

Taktéž můžeme pracovat i s optimistickým odhadem chyby na testovacím souboru, který se značí jako $e'(t) = 1 - OA_{\text{test}}$, díky kterému získáme objektivnější výsledky.

Přesnost regresního stromu se určuje stejně jako v lineární regresi, a to za pomoci koeficientu determinace R^2. Ten je definován jako podíl variability závisle proměnné Y ku celkové variabilitě proměnné Y.

$$R^2 \quad = \quad 1 - \frac{\sum_{i=1}^{n} (y_i - \widehat{y_i})^2}{\sum_{i=1}^{n} (y_i - \overline{y})^2}, \tag{6}$$

kde $\widehat{y_i} = \overline{y_t}$ značí průměr v konkrétních terminálních uzlech. Pro uzel t se odchylka od jeho průměru počítá pro pozorování y_i, které je zařazeno do daného terminálního uzlu.
Koeficient determinace R^2 může nabývat hodnot mezi 0 a 1 včetně. Je-li hodnota koeficientu determinace $R^2 = 1$, pak je pomocí stromu vysvětlena veškerá variabilita, což znamená, že pozorované hodnoty y_i odpovídají predikovaným hodnotám $\widehat{y_i}$.
U regresního stromu můžeme také vypočítat jeho chybu, a to jak pro trénovací, tak i pro testovací soubor. Chyba pro trénovací soubor se vypočítá jako $e(t) = 1 - R^2_{\text{tren}}$. Obdobně se vypočítá chyba pro testovací soubor $e'(t) = 1 - R^2_{test}$.

3.3.9 Výhody rozhodovacích stromů CART

Jedna z hlavních výhod použití rozhodovacích stromů je, že není potřeba využívat transformace proměnných, jelikož na rozložení dat stromy nekladou nároky. Rozhodovací stromy jsou tedy vhodné i pro velký počet proměnných, u kterých nezáleží na jejich typu. Rozhodovací stromy lze snadno zaznamenávat pomocí grafů se stromovou strukturou. Pomocí algoritmu tvorby stromu můžeme včas odhalit odlehlé hodnoty pomocí křížové validace. Jelikož růst stromu je hierarchický a pro dělení se vždy vybere pouze jeden prediktor, je možno použít korelované prediktory. Koeficient determinace R^2 u regresního stromu je možno srovnávat s R^2 u jiných regresních technik, také procento správně klasifikovaných pozorování lze srovnávat s výstupy jiných klasifikačních metod. Pro klasifikaci nových případů se jedná o velmi rychlou metodu, která je ovšem vhodná nejen pro klasifikaci, ale i pro regresi, ovšem s jistými omezeními.

3.3.10 Nevýhody rozhodovacích stromů CART

Jednou z hlavních nevýhod rozhodovacích stromů je jejich vysoká nestabilita (kterou je ovšem možno řešit pomocí lesů). Nestabilitou stromu myslíme to, že jeho tvar je velmi závislý na datech. I malá změna dat může způsobit velké změny v rozhodovacích pravidlech uvnitř uzlů, které mohou způsobit až změnu výsledných klasifikací a predikcí. Právě kvůli této nestabilitě stromu je potřeba opatrnosti při interpretaci stromu. Měření přesnosti stromu výrazně závisí na mechanismu křížové validace a dalších parametrech při validaci daného modelu v procesu

učení (jedná se např. o pravidla pro zastavení růstu stromu). Rozhodovací stromy také nejsou vhodné pro příliš malý počet vzorků a taktéž nejsou vhodné pro velký počet kategorií závisle proměnné. Při vytváření stromů potřebujeme využívat zkušenosti s nastavením parametrů v procesu validace, což je celkem subjektivní.

3.4 Rozhodovací stromy v OpenCV

Nyní si popíšeme jak fungují rozhodovací stromy v praktickém použití pomocí OpenCV. Rozhodovací stromy jsou pro verzi, kterou jsme použili, reprezentovány třídou CvDTree, pomocí které jsme si vytvořili jednoduchý binární rozhodovací strom. Pro klasifikační rozhodovací stromy každý listový uzel obsahuje označení třídy, do které objekt patří, kdežto pro regresní rozhodovací stromy tyto uzly neobsahují pouze označení třídy, nýbrž i nějakou konstantu.

Naším cílem je ze vstupního vektoru příznaků, který získáme pomocí použití metody HOG, získat odezvu na tento vektor pomocí rozhodovacího stromu a to formou třídy, do které zkoumaný objekt, který tímto vektorem reprezentujeme, patří. Proces samozřejmě začíná v kořeni stromu a pokračuje do konkrétních dceřiných uzlů až k listu. Jelikož máme binární strom, tak se v každém uzlu musíme rozhodnout, zda pokračovat levým nebo pravým dceřiným uzlem, a to na základě hodnoty konkrétní proměnné, jejíž index je uložen ve zkoumaném uzlu rozhodovacího stromu. Pro CvDTree jsou možné dva druhy proměnných - uspořádané proměnné a kategorické proměnné.

Pro první typ proměnných je každá hodnota proměnné porovnána s nějakým prahem, který je uložen ve zkoumaném uzlu stromu, a je-li tato hodnota menší než práh v uzlu, pak pokračujeme v rozhodovacím stromě směrem vlevo, jinak směrem vpravo, tedy pokračujeme dále pravým dceřiným uzlem (případně vstupujeme do listového uzlu). Jako příklad si můžeme vzít již dříve zmíněný rozhodovací strom znázorněn na obr. 9 s uzlem, který obsahuje rozhodnutí pro náš dříve zmíněný objekt „nový potkan“, a to konkrétně ohledně váhy potkana. Porovnávání v uzlu bychom provedli s hodnotou $100\,g$, kdy v případě, že váha potkana by byla méně než $100\,g$, pokračovali bychom v rozhodovacím stromu do levého dceřiného uzlu určující, že se nejedná o dospělého jedince, a v případě, že by váha nebyla menší než $100\,g$, pokračovali bychom do pravého dceřiného uzlu, který by určoval, že se jedná o dospělého jedince.

Pro druhý typ proměnných (kategorické proměnné) se hodnota diskrétní proměnné testuje k určité množině hodnot uložené ve zkoumaném uzlu rozhodovacího stromu z omezené sady hodnot, ze které může proměnná nabývat. I zde platí pro kladný případ, že postupujeme do levého dceřiného uzlu, jinak pokračujeme pravým dceřiným uzlem. Příklad si znovu můžeme ukázat na dříve zmíněném rozhodovacím stromu znázorněném na obr. 8 na ukázkovém objektu „nový potkan“ a jeho atributu „barva srsti“, kde si nadefinujme, že pro hodnoty „šedá“ a „černá“ by rozhodování pokračovalo levým dceřiným uzlem a pro jiné hodnoty bychom pokračovali do

pravého dceřiného uzlu.

V rozhodovacím stromě tedy pro každý nelistový uzel máme dvojici (variable_index , decision_rule (threshold / subset)) a jakmile po celém procesu rozhodovacího stromu narazíme na listový uzel, pak je hodnota, která se nachází v daném listovém uzlu, použita jako výsledek rozhodovacího stromu a tedy jako výstup celého procesu rozhodovacího stromu.

Nyní můžeme přejít k trénování rozhodovacích stromů pomocí OpenCV. Ty se zde vytvářejí rekurzivně z kořenového uzlu, kde jsou použity všechny příznakové vektory s informacemi do jakých tříd patří. Na základě určitého kritéria, pro klasifikaci se jedná o Gini a pro regresi součet kvadratických chyb, je nalezeno v každém uzlu rozhodovacího stromu rozhodovací pravidlo, tzv. nejlepší primární rozdělení. Jakmile dojde k rozdělení na kořenovém uzlu, rekurzivně se opakuje rozdělení na levém i pravém dceřiném uzlu. Tento proces se může zastavit na kterémkoli uzlu, a to buď díky dosáhnutí maximální nadefinované hodnoty hloubky stromu, nebo jestliže je počet trénovacích objektů v uzle menší než nějaký nadefinovaný práh (tedy nedostatečně reprezentativní pro další větvení z daného uzlu), nebo pokud náleží všechny objekty v uzlu do stejné třídy pro kategorický případ a je příliš malá variace pro regresi, a jako poslední poslední možnost si zmíníme zastavení růstu stromu ve chvíli, kdy nejlepší možné rozdělení již nepřinese stromu žádné zlepšení ve srovnání s náhodným výběrem. Strom také můžeme po jeho sestavení v případě potřeby prořezat pomocí křížové validace.

V OpenCV máme pro rozhodovací stromy různá nastavení. Společně pro SVM a náhodné lesy můžeme mít nastavení kritéria pro ukončení iterativního algoritmu, tedy CvTermCriteria, které si tedy zmíníme již v této kapitole. Parametr epsilon nám udává požadovanou přesnost algoritmu a parametr max_iter maximální počet iterací tohoto algoritmu. Pro parametr type máme tři možnosti. První možnost je nastavení hodnoty na CV_TERMCRIT_ITER, která zajistí zastavení algoritmu po dosažení maximálního počtu iterací. Druhou možností je nastavení hodnoty na CV_TERMCRIT_EPS, která zajistí zastavení algoritmu po dosažení nižší přesnosti než námi nadefinované v předchozím kroku. Poslední možnost je kombinace CV_TERMCRIT_ITER+ CV_TERMCRIT_EPS, která algoritmus zastaví buď po dosažení maximálního počtu iterací, nebo jakmile algoritmus dosáhne nižší přesnosti než je nadefinována, podle toho, která z možností nastane dříve.

```
CvTermCriteria crit;
   crit.epsilon = hodnota1;
   crit.max_iter = hodnota2;
   crit.type = hodnota3;
```

Ukázka 1: Kritérium pro ukončení iterativního algoritmu.

Když jsme si uvedli základní nastavení pro algoritmus rozhodovacího stromu i SVM, můžeme se posunout na nastavení typické pouze pro rozhodovací stromy v OpenCV. V této části si tedy představíme strukturu CvDTreeParams, která obsahuje veškeré potřebné parametry pro trénování rozhodovacího stromu.

Jako první parametr si uveďme max_depth, který nám udává maximální možnou dosažitelnou hloubku stromu. Parametr min_sample_count určuje minimální počet vzorků v uzlu stromu pro další dělení stromu. Další možnost nastavení nám dává parametr regression_accuracy , který zastaví dělení stromu v případě, že absolutní hodnota rozdílu odhadované hodnoty v uzlu stromu a hodnot trénovacích dat v daném uzle je menší než nadefinovaná hodnota parametru. Parametr use_surrogates určuje možnost náhradních dělení uzlu stromu. Takováto možnost se může hodit v případech, kdy nám chybí nějaké příznaky (např. v šeru barva objektu) a procedura se zasekne na daném uzlu (např. když se má dále dělit podle určité barvy). V těchto případech se tedy bude moct strom na daném uzlu dále dělit pomocí jiných proměnných s velmi podobnými výsledky. Dalším parametrem pro trénování rozhodovacího stromu pomocí OpenCV je max_categories, omezení kategorické proměnné k nalezení vhodného rozdělení uzlu stromu na $K \leq$ max_categories. Pro případ regresních klasifikací a klasifikací obsahujících pouze dvě třídy není tento parametr využíván. Následující parametr je cv_folds , který jestliže je větší než 1, pak se strom prořeže pomocí K-násobné křížové validace, kde K značí hodnotu parametru cv_folds větší než 1. S tímto parametrem souvisí další, a to use_1se_rule , který v případě hodnoty true bude přísněji prořezávat náš rozhodovací strom. Parametr truncate_pruned_tree udává možnost fyzického odstranění ořezaných větví rozhodovacího stromu (pro hodnotu true). V opačném případě, kdy se větví nechceme nevratně zbavovat, k nim můžeme přistupovat pomocí parametru pruned_tree_idx . Posledním parametrem je priors , pole možností tříd umožňující vyladění preferencí rozhodovacího stromu směrem k určité třídě. Můžeme si jej představit jako váhu predikce kategorií.

```
CvDTreeParams params;
   params.max_depth = hodnota1;
   params.min_sample_count = hodnota2;
   params.regression_accuracy = hodnota3;
   params.use_surrogates = hodnota4;
   params.max_categories = hodnota5;
   params.cv_folds = hodnota6;
   params.use_1se_rule = hodnota7;
   params.truncate_pruned_tree = hodnota8;
   params.priors = hodnota9;
```

Ukázka 2: Nastavení parametrů pro trénovaní rozhodovacího stromu.

Praktická ukázka implementace rozhodovacích stromů pomocí OpenCV je součástí programu této práce a její výsledky můžeme vidět v kapitole 6, stejně jako sadu experimentů k otestování HOG algoritmu na rozhodovacích stromech pro detekci obsazenosti parkovacích míst na parkovišti.

4 Support Vector Machine (SVM)

Existuje mnoho metod strojového učení, které zvládají jednoduché a efektivní učící algoritmy, nicméně se dokáží naučit pouze lineární oddělovače k řešení úlohy nalezení hranice oddělující dané třídy ve vstupním prostoru. Na druhé straně máme také metody, které obecné nelineární funkce zvládají, nicméně u nich bývá obtížné učení. Jako alternativu k těmto metodám si představíme druhou metodu pro učení pomocí příznaků a pro následné rozpoznávání objektů. Metoda SVM („Support Vector Machine“, „Support Vector Network“, „Podpůrné vektory“) patří mezi metody strojového učení a implementuje myšlenku nelineárního mapování vstupních vektorů do vícerozměrného prostoru příznaků, ve kterém vytvoříme lineární rozhodovací plochu. Speciální vlastnosti rozhodovací plochy zajišťují schopnost vysoké generalizace strojového učení.

4.1 Počáteční vývoj

V roce 1936 byl navrhnut R. A. Fisherem [15] první algoritmus pro rozpoznávání vzorů. Fisher vzal v úvahu model dvou populací s normálním rozdělením, $N(\mathbf{m_1}, \sum_1)$ a $N(\mathbf{m_2}, \sum_2)$ n-rozměrných vektorů $\mathbf{x}$, s vektory $\mathbf{m_1}$ a $\mathbf{m_2}$ a kovarianci matic $\sum_1$ a $\sum_2$ a ukázal, že optimálním řešením (Bayesian) je kvadratická rozhodovací funkce:

$$F_{\text{sq}}(\mathbf{x}) \quad = \quad \text{sign}\left[\frac{1}{2}(\mathbf{x}-\mathbf{m}_1)^T \sum_1^{-1}(\mathbf{x}-\mathbf{m}_1) - \frac{1}{2}(\mathbf{x}-\mathbf{m}_2)^T \sum_2^{-1}(\mathbf{x}-\mathbf{m}_2) + \ln\frac{|\sum_2|}{|\sum_1|}\right] \tag{7}$$

Pokud bychom vzali v úvahu případ, kdy $\sum_1 = \sum_2 = \sum$, pak se kvadratická rozhodovací funkce změní na lineární funkci:

$$F_{\text{lin}}(\mathbf{x}) \quad = \quad \text{sign}\left[(\mathbf{m}_1-\mathbf{m}_2)^T \sum^{-1}\mathbf{x} - \frac{1}{2}\left(\mathbf{m}_1^T \sum^{-1}\mathbf{m}_1 - \mathbf{m}_2^T \sum^{-1}\mathbf{m}_2\right)\right]. \tag{8}$$

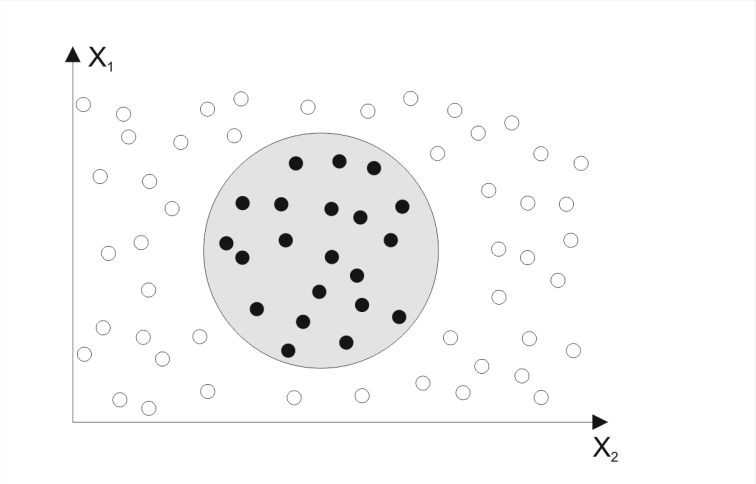
Pro odhad kvadratické rozhodovací funkce musíme určit $\frac{n(n+3)}{2}$ parametrů, pro lineární funkci by to bylo pouze n parametrů k určení. Pro malý počet pozorování však byl tento postup nespolehlivý. Jako další možnost doporučil Fisher lineární rozhodovací funkci pro případ dvou distribucí, které nemají normální rozdělení. Algoritmy pro rozpoznávání vzorů byly tedy již od samých počátků spojeny s konstrukcí lineární rozhodovací plochy.

Jako další část vývoje si můžeme uvést Rosenblatta [11] a jeho výzkum perceptronů neuronových sítí. Perceptrony se skládaly z propojených neuronů, kde každý neuron implementoval oddělující nadrovinu, tedy perceptron jako celek implementoval po částech lineárně separující se plochu.

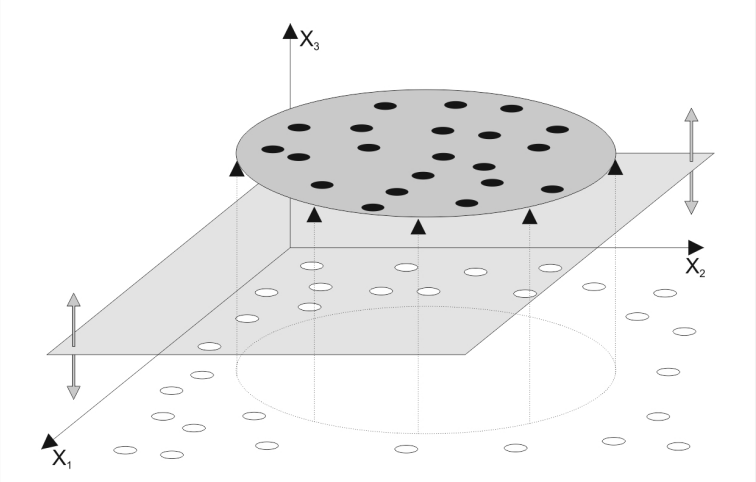
4.2 Teorie SVM

Jak jsme zmínili v úvodu této kapitoly, SVM implementuje mapování vstupních vektorů do vícerozměrného prostoru příznaků, nazvěme jej Z, prostřednictvím nějakého nelineárního mapování. V tomto prostoru příznaků vytvoříme lineární rozhodovací plochu se speciálními vlastnostmi zajišťujícími schopnost vysoké generalizace strojového učení.

Na obr. 11 můžeme vidět zobrazení dvou tříd, a to nelineárně oddělené (pomocí kružnice). Pokud bychom ovšem přidali další dimenzi, pak bychom měli možnost body jedné třídy posunout o nově přidanou souřadnici a poté bychom již mohli vytvořit rovinu rovnoběžnou s ostatními osami a touto rovinou je od sebe oddělit. Tento proces jsme si znázornili na obr. 12.

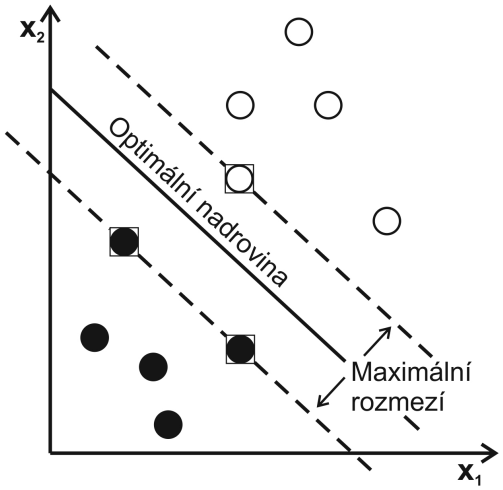


Obrázek 11: Příklad nelineárně oddělených tříd pomocí kružnice



Obrázek 12: Příklad lineárního oddělení tříd pomocí nově přidané dimenze

Naším úkolem tedy je nalezení nejvhodnějšího umístění lineární hranice, tedy tak, aby byla co nejefektivnější pro kategorizaci vstupních dat. Jak jsme si ukázali na obr. 12, SVM dokáže nejen lineární oddělení tříd, ale i složitější nelineární funkce, tudíž pomocí této metody můžeme oddělovat i lineárně neseparabilní třídy. Na obr. 11 a 12 jsme mohli vidět jednoduchý příklad převodu vstupního prostoru na více-dimenzionální prostor, ve kterém lze snadno dané třídy lineárně od sebe oddělit. Na prvním obrázku jsou vidět příznaky, které od sebe nelze oddělit lineárně, ale pouze pomocí kružnice, kdežto na následujícím obrázku máme tento prostor rozšířen o další dimenzi a existuje oddělující nadrovina, která separuje jednotlivé třídy.



Obrázek 13: Ukázka oddělitelného problému ve dvourozměrném prostoru

Při použití SVM nám ale vznikají dva problémy, a to

- jak nalézt nejvhodnější rozdělující nadrovinu, jelikož počet rozměrů prostoru příznaků bude velký,
- a také jak výpočetně pracovat s vícerozměrovými prostory.

První část problému se řeší pomocí *optimálních nadrovin* pro oddělitelné třídy. Optimální nadrovinou rozumíme lineární rozhodovací funkci s maximálním rozmezím (šířkou, pásmem) mezi vektory dvou tříd, ukázku můžeme vidět na obr. 13.

4.3 Optimální nadrovina

K sestavení optimální nadroviny budeme potřebovat malé množství trénovacích dat, nazveme je *podpůrné vektory* („support vectors“), které nám budou určovat zmíněné rozmezí. Jestliže by se nám podařilo trénovací vektory oddělit bez chyb optimální nadrovinou, pak by očekávaná

hodnota pravděpodobnosti dopuštění se chyby na testovacím příkladu byla

$$E\left[\Pr\left(\text{chyba}\right)\right] \quad \leq \quad \frac{E\left[\text{pocet podpurnych vektoru}\right]}{\text{pocet trenovacich vektoru}}. \tag{9}$$

Označme si

$$\mathbf{w}_0 \cdot \mathbf{z} + b_0 \quad = \quad 0 \tag{10}$$

jako optimální nadrovinu v prostoru příznaků, ve které si váhy $\mathbf{w}_0$ můžeme zapsat jako lineární kombinaci podpůrných vektorů

$$\mathbf{w}_0 \quad = \quad \sum_{\text{supp vec}} \alpha_i \mathbf{z}_i. \tag{11}$$

Lineární rozhodovací funkce $l(\mathbf{z})$ v prostoru příznaků pak bude mít podobu

$$l(\mathbf{z}) \quad = \quad \text{sign}\left(\sum_{\text{supp vec}} \alpha_i \mathbf{z}_i \cdot \mathbf{z} + b_0\right) \tag{12}$$

a $\mathbf{z}_i \cdot \mathbf{z}$ nám bude značit skalární součin mezi podpůrnými vektory $\mathbf{z}_i$ a vektorem $\mathbf{z}$ v prostoru příznaků.

4.3.1 Algoritmus optimální nadroviny

Sada označených trénovacích vzorů

$$\left(y_1, \mathbf{x}_1\right), \ldots, \left(y_l, \mathbf{x}_l\right), \qquad y_i \in \{-1, 1\} \tag{13}$$

je lineárně oddělitelná, jestliže existuje vektor $\mathbf{w}$ a skalár b takový, že nerovnosti

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i + b \quad &\geq \quad 1, \qquad \text{pokud} \quad y_i = 1, \\ \mathbf{w} \cdot \mathbf{x}_i + b \quad &\leq \quad -1, \qquad \text{pokud} \quad y_i = -1, \end{aligned} \tag{14}$$

platí pro všechny prvky trénovací sady (13). Nerovnosti (14) můžeme také přepsat jako:

$$y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right) \quad \geq \quad 1, \qquad i = 1, \ldots, l. \tag{15}$$

Optimální nadrovinu si nyní nadefinujme jako

$$\mathbf{w}_0 \cdot \mathbf{x} + b_0 \quad = \quad 0. \tag{16}$$

Tato nadrovina bude oddělovat trénovací data s maximálním rozmezím a určovat směr $\frac{\mathbf{w}}{|\mathbf{w}|}$, ve kterém bude rozdíl mezi projekcemi trénovacích vektorů dvou rozdílných tříd maximální (obr. 13).

Nyní si můžeme nadefinovat vzdálenost $\rho(\mathbf{w}, b)$ jako

$$\rho(\mathbf{w}, b) \quad = \quad \min_{\{x:y=1\}} \frac{\mathbf{x} \cdot \mathbf{w}}{|\mathbf{w}|} - \max_{\{x:y=-1\}} \frac{\mathbf{x} \cdot \mathbf{w}}{|\mathbf{w}|}, \tag{17}$$

kde nám bude optimální nadrovina $(\mathbf{w}_0, b_0)$ maximalizovat vzdálenost $\rho(\mathbf{w}, b)$. Z (17) a (15) můžeme také odvodit

$$\rho(\mathbf{w}_0, b_0) \quad = \quad \frac{2}{|\mathbf{w}_0|} = \frac{2}{\sqrt{\mathbf{w}_0 \cdot \mathbf{w}_0}}. \tag{18}$$

Máme-li vektory $\mathbf{x}_i$, pro které platí rovnost

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \quad = \quad 1, \tag{19}$$

pak se jedná o vektory podpůrné. Vektor $\mathbf{w}_0$ určující optimální nadrovinu můžeme nyní zapsán jako lineární kombinaci trénovacích vektorů

$$\mathbf{w}_0 \quad = \quad \sum_{i=1}^{l} y_i \alpha_i^0 \mathbf{x}_i, \qquad \alpha_i^0 \geq 0 \tag{20}$$

Pro nalezení vektoru $\mathbf{\Lambda}_0^T = (\alpha_1^0, \ldots, \alpha_l^0)$ o parametrech α_i musíme vyřešit

$$W(\mathbf{\Lambda}) \quad = \quad \mathbf{\Lambda}^T \mathbf{1} - \frac{1}{2} \mathbf{\Lambda}^T \mathbf{D} \mathbf{\Lambda} \tag{21}$$

za pomocí podmínek

$$\mathbf{\Lambda} \quad \geq \quad 0, \tag{22}$$

$$\mathbf{\Lambda}^T \mathbf{Y} \quad = \quad 0 \tag{23}$$

a kde $\mathbf{1}^T = (1, \ldots, l)$ značí l-rozměrný jednotkový vektor, $\mathbf{Y}^T = (y_1, \ldots, y_l)$ značí l-rozměrný vektor označení a D značí symetrickou $l \times l$ matici s prvky

$$D_{ij} \quad = \quad y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j, \qquad i, j = 1, \ldots, l. \tag{24}$$

Nerovnost (22) nám popisuje nezáporný kvadrant, a proto musíme maximalizovat kvadratickou formu (21) v nezáporném kvadrantu pomocí podmínek (23).

V případě oddělení trénovacích dat bez chyb platí

$$W(\mathbf{\Lambda}_0) = \frac{2}{\rho_0^2}. \tag{25}$$

Jestliže pro některé $\mathbf{\Lambda}_*$ a vysokou konstantu W_0 platí nerovnost

$$W(\mathbf{\Lambda}_*) > W_0, \tag{26}$$

pak podle toho můžeme tvrdit, že všechny nadroviny separující trénovací data mají rozmezí

$$\rho < \sqrt{\frac{2}{W_0}}. \tag{27}$$

Jestliže však nemůže být trénovací sada rozdělena nadrovinou, stane se rozmezí libovolně malé, což zapříčiní změnu hodnoty funkce $W(\mathbf{\Lambda})$ na libovolně velkou.

Chceme-li získat maximální rozmezí, maximalizujeme funkci (21) podle podmínek (22) a (23). Rozdělíme trénovací data na části s rozumně malým množstvím trénovacích vektorů v každé z těchto částí a pro první část vypočítáme funkci (21). Výsledkem může být buď nemožnost rozdělení této části dat pomocí nadroviny (pak můžeme říct, že celá sada dat nemůže být rozdělena), nebo se nám podaří najít optimální nadrovinu separující první část dat.

Mějme vektor $\mathbf{\Lambda}_1$ maximalizující funkci (21) v takovémto případě rozdělení první části. Vytvoříme novou sadu trénovacích dat obsahující podpůrné vektory z první části trénovacích dat a vektory druhé části, které nesplňují podmínku (15), kde $\mathbf{w}$ je určen pomocí $\mathbf{\Lambda}_1$. Pro tuto sadu vytvoříme funkci $W_2(\mathbf{\Lambda})$ a maximalizujeme v $\mathbf{\Lambda}_2$. Tímto postupem můžeme vytvořit optimální nadrovinu pro celý náš soubor dat $\mathbf{\Lambda}_* = \mathbf{\Lambda}_0$. Během tohoto procesu se ovšem zvyšuje hodnota funkce $W(\mathbf{\Lambda})$, jelikož zvyšujeme počet trénovacích vektorů v rámci optimalizace, což vede k čím dál tím menšímu rozdělení mezi dvěma třídami.

4.3.2 Jemné rozmezí nadroviny

V předchozí kapitole jsme si uvedli příklad, kdy jsou data rozdělena bezchybně. Ne vždy však je možno je takto rozdělit, a proto si nyní uvedeme případ, kdy data takto rozdělena být nemohou. Naším cílem bude oddělit trénovací sadu s minimálním množstvím chyb.

Zaveďme si proměnnou $\xi_i \geq 0$, $i = 1, \ldots, l$ a minimalizujme funkci

$$\Phi(\xi) = \sum_{i=1}^{l} \xi_i^{\sigma} \tag{28}$$

pro nějaké malé $\sigma > 0$ za podmínek

$$y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b \right) \geq 1 - \xi_i, \qquad i = 1, \ldots, l, \tag{29}$$

$$\xi_i \geq 0, \qquad i = 1, \ldots, l. \tag{30}$$

Budeme-li mít dostatečně malé σ, pak bude funkce (28) označovat počet trénovacích chyb.

Minimalizací (28) můžeme získat minimální podmnožinu trénovacích chyb:

$$\left(y_{i_1}, \mathbf{x}_{i_1} \right), \ldots, \left(y_{i_k}, \mathbf{x}_{i_k} \right). \tag{31}$$

Jestliže vyloučíme tato data z trénovací sady, pak můžeme oddělit zbývající část trénovací sady bez chyb. K oddělení zbývající části trénovacích dat můžeme vytvořit optimální oddělující nadrovinu.

Tento postup můžeme vyjádřit jako minimalizaci funkce

$$\frac{1}{2}\mathbf{w}^2 + CF\left(\sum_{i=1}^{l} \xi_i^{\sigma} \right) \tag{32}$$

za podmínek (29) a (30) ($F(u)$ nám bude značit monotónní konvexní funkci a C konstantu).

Pro dostatečně velké C a dostatečně malé σ nám vektor $\mathbf{w}_0$ a konstanta b_0 určují nadrovinu, která minimalizuje počet chyb na trénovací sadě.

Konstrukce nadroviny minimalizující počet chyb na trénovací sadě je obecný NP-úplný problém. Jestliže bychom se chtěli vyhnout NP-úplnosti, mohli bychom určit $\sigma = 1$ (nejmenší hodnotu σ, pro kterou optimalizační problém (21) má jediné řešení). Jestliže mohou být trénovací data oddělena bez chyb, pak se vytvořená nadrovina shoduje s optimálním rozmezím nadroviny.

V porovnání s případem $\sigma < 1$ existuje efektivní metoda pro nalezení řešení (32) v případě $\sigma = 1$. Takovéto řešení nazýváme jemné rozmezí nadroviny.

4.3.3 Metoda konvoluce skalárního součinu v prostoru příznaků

Algoritmy, které jsme si popsali v předchozích podkapitolách vytvářejí nadroviny ve vstupním prostoru. K takovémuto vytvoření nadroviny v prostoru příznaků musíme ale nejdříve transformovat n-rozměrný vstupní vektor $\mathbf{x}$ na N-rozměrný vektor příznaků napříč výběru M-

rozměrného vektoru funkce ϕ:

$$\phi \quad = \quad \mathbb{R}^n \to \mathbb{R}^N. \tag{33}$$

Můžeme poté vytvořit N-rozměrný lineární oddělovač $\mathbf{w}$ a zkreslení b, a to pro sadu transformovaných vektorů

$$\phi(\mathbf{x}_i) \quad = \quad \phi_1(\mathbf{x}_i), \phi_2(\mathbf{x}_i), \ldots, \phi_N(\mathbf{x}_i), \qquad i = 1, \ldots, l. \tag{34}$$

Klasifikaci neznámého vektoru $\mathbf{x}$ získáme pomocí transformace vektoru do rozdělujícího prostoru ($\mathbf{x} \longmapsto \phi(\mathbf{x})$) a poté pomocí sign funkce

$$f(\mathbf{x}) \quad = \quad \mathbf{w} \cdot \phi(\mathbf{x}) + b. \tag{35}$$

Klasifikační metodu vektoru $\mathbf{w}$ můžeme zapsat jako lineární kombinaci podpůrných vektorů v prostoru příznaků

$$\mathbf{w} \quad = \quad \sum_{i=1}^{l} y_i \alpha_i \phi(\mathbf{x}_i). \tag{36}$$

Vytvoření SVM vychází z uvážení obecných forem skalárního součinu v Hilbertově prostoru ([13]):

$$\phi(\mathbf{u}) \cdot \phi(\mathbf{v}) \quad \equiv \quad K(\mathbf{u}, \mathbf{v}). \tag{37}$$

Vzhledem k Hilbert-Schmidt teorii ([14]) může být jakákoli symetrická funkce $K(\mathbf{u}, \mathbf{v})$ ($K(\mathbf{u}, \mathbf{v}) \in L_2$) rozšířena na

$$K(\mathbf{u}, \mathbf{v}) \quad = \quad \sum_{i=1}^{\infty} \mathbf{\Lambda}_i \phi_i(\mathbf{u}) \cdot \phi_i(\mathbf{v}), \tag{38}$$

kde $\mathbf{\Lambda}_i \in \mathbb{R}$ a ϕ_i jsou vlastní hodnoty a vlastní funkce

$$\int K(\mathbf{u}, \mathbf{v}) \phi_i(\mathbf{u}) \, d\mathbf{u} \quad = \quad \mathbf{\Lambda}_i \phi_i(\mathbf{v}) \tag{39}$$

integrálního operátoru definovaného jádrem $K(\mathbf{u}, \mathbf{v})$. Pro skalární součin v prostoru příznaků musí být vlastní hodnoty v (38) kladné. Pro zajištění kladnosti koeficientů je zapotřebí, aby podmínka

$$\int\int K(\mathbf{u}, \mathbf{v}) g(\mathbf{u}) g(\mathbf{v}) \, \mathrm{d}\mathbf{u}\mathrm{d}\mathbf{v} \quad > \quad 0 \tag{40}$$

byla splněna pro všechna g taková, že

$$\int g^2(\mathbf{u})\,\mathrm{d}\mathbf{u} \;<\; \infty \tag{41}$$

Jestliže funkce tyto podmínky splňují, pak mohou být použity jako skalární součiny.

Konvoluce skalárního součinu v prostoru příznaků může být ale dána jakoukoli funkcí splňující výše zmíněné podmínky. Pro konstrukci polynomického klasifikátoru stupně d v n-rozměrném prostoru pak můžeme použít funkci

$$K(\mathbf{u},\mathbf{v}) \;=\; (\mathbf{u}\cdot\mathbf{v}+\mathbf{1})^d. \tag{42}$$

Použitím různých skalárních součinů $K(\mathbf{u},\mathbf{v})$ můžeme získat různé učící stroje s různými typy rozhodovacích ploch. Rozhodovací plocha takovýchto učících strojů pak je definována jako

$$f(\mathbf{x}) \;=\; \sum_{i=1}^{l} y_i \alpha_i K(\mathbf{x},\mathbf{x}_i), \tag{43}$$

kde $\mathbf{x}_i$ značí obraz podpůrného vektoru ve vstupním prostoru a α_i váhu podpůrného vektoru v prostoru příznaků.

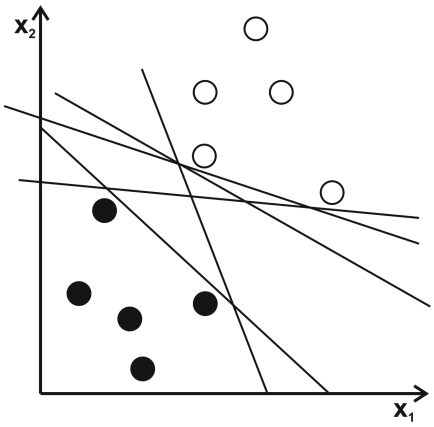
Pro nalezení vektorů $\mathbf{x}_i$ a vah α_i můžeme postupovat stejně jako pro klasifikátor původního optimálního rozmezí nebo pro klasifikátor jemného rozmezí. Rozdíl je v použití matice

$$D_{i,j} \;=\; y_i y_j K(\mathbf{x}_i,\mathbf{x}_j), \qquad i,j=1,\ldots,l. \tag{44}$$

SVM nám umožňuje řešení pomocí optimálních nadrovin (rozšíření výsledného vektoru na podpůrné vektory), konvoluci skalárního součinu (rozšíření výsledné plochy z lineárních na nelineární) a jemné rozmezí (poskytnutí chyb na trénovací sadě) a z SVM se tak stal velmi silný učící stroj.

4.4 SVM v OpenCV

Nyní si popišme jak fungují SVM v praktickém použití pomocí OpenCV. Support Vector Machines jsou pro verzi, kterou jsme použili, reprezentovány třídou CvSVM, pomocí které jsme si vytvořili jednoduchý binární klasifikátor SVM. Na obr. 14 můžeme vidět jednoduchou ukázku lineární separability bodů dvou tříd ve $2D$ prostoru. Jak jsme již zmínili v teoretické části, existuje mnoho možných (v tomto případě) přímek, které jsou řešením. Naším cílem je najít nejvhodnější z nich, např. jestliže by přímka byla příliš blízko bodům, pak by model byl velmi citlivý na šum a negeneralizoval tak dobře, jako výhodnější volba separující přímky. Snažíme se tedy nalézt takovou přímku, která bude od všech bodů v prostoru co nejdále.



Obrázek 14: Ukázka možných oddělujících přímek ve $2D$ prostoru dvou tříd

Obecně se v OpenCV pro SVM snažíme nalézt optimální nadrovinu, která bude mít největší minimální vzdálenost ke všem bodům trénovací sady příznaků, jak jsme mohli vidět na obr. 13.

Pro jednoduchost si znovu nadefinujme nadrovinu jako funkci

$$f(x) \quad = \quad \beta_0 + \beta^T x, \tag{45}$$

kde β značí váhový vektor, β_0 značí práh a x značí podpůrné vektory. Budeme-li mít prostor ve $2D$, budou trénovací data rozdělena pomocí přímky (jak jsme viděli v začátku této kapitoly), pro $3D$ se bude jednat o rovinu. Jak můžeme vidět z rovnice (45), optimální nadrovina může být reprezentována nekonečně mnoha způsoby, a z tohoto důvodu si můžeme uvést vztah

$$\left|\beta_0 + \beta^T x\right| \quad = \quad 1, \tag{46}$$

ve kterém si jasně nadefinujeme konkrétní nadrovinu z nekonečného počtu možností. Vzdálenost mezi bodem x a nadrovinou pak máme definovanou jako

$$\text{vzdalenost} \quad = \quad \frac{\left|\beta_0 + \beta^T x\right|}{||\beta||}, \tag{47}$$

nebo také

$$\text{vzdalenost} \quad = \quad \frac{\left|\beta_0 + \beta^T x\right|}{||\beta||} = \frac{1}{||\beta||}. \tag{48}$$

Rozmezí (margin) M je, jak můžeme vidět na obr. 13, pak dvojnásobkem nejmenší vzdále-

nosti, tedy

$$M = \frac{2}{||\beta||}. \tag{49}$$

Nyní si můžeme uvést konečný vztah, a to pro maximalizaci M pomocí minimalizace funkce $L(\beta)$ za určitých podmínek, konkrétně

$$\min_{\beta,\beta_0} L(\beta) = \frac{1}{2}||\beta||^2, \qquad y_i\left(\beta_0 + \beta^T x_i\right) \geq 1 \forall i, \tag{50}$$

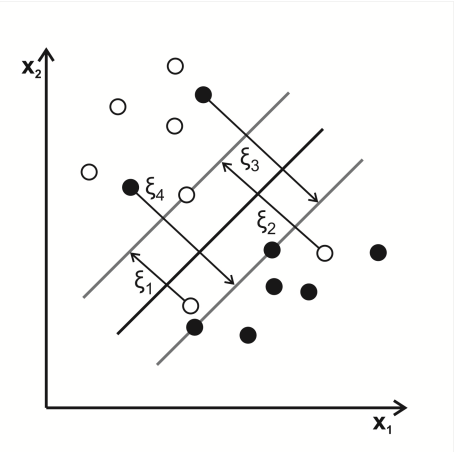
kde y_i značí každou třídu trénovacích dat. Tento problém Lagrangeovy optimalizace může být vyřešen pomocí Lagrangeova násobitele pro získání váhového vektoru β a prahu β_0 pro optimální nadrovinu.

Pro nelineárně separovatelná trénovací data musíme počítat s tím, že nalezená nadrovina bude klasifikovat data s chybami a musíme s touto chybou pracovat jako s novou proměnnou. Připomeňme si optimalizační problém z (50), který ovšem neřeší špatně klasifikovaná data. Naším cílem je nalezení takové nadroviny, která bude mít co největší rozmezí a přitom bude klasifikovat data správně s co nejmenším počtem chyb. Jako vhodné řešení si můžeme uvést

$$\min ||\beta||^2 + C * \xi, \tag{51}$$

kde C je nějaká konstanta a ξ je vzdálenost špatně klasifikovaných dat od jejich správného umístění (klasifikace). Ukázku můžeme vidět na obr. 15. Mějme ξ_i definující tyto vzdálenosti a tedy

$$\min_{\beta,\beta_0} L(\beta) = ||\beta||^2 + C\sum_i \xi_i, \qquad y_i\left(\beta^T x_i + \beta_0\right) \geq 1 - \xi_i, \text{ a } \xi_i \geq 0 \forall i, \tag{52}$$



Obrázek 15: Ukázka vzdáleností špatně klasifikovaných dat od jejich správného umístění

Volba parametru C není pevně daná, nicméně větší hodnoty C nám dají méně špatně klasifikovaných dat, ale s menším pásmem, naopak menší hodnoty C nám vytvoří větší pásmo, ale více chybných klasifikací.

V OpenCV můžeme pro SVM používat různá nastavení. Jak jsme již zmínili, společné pro SVM i rozhodovací stromy je nastavení kritéria pro ukončení iterativního algoritmu, tedy CvTermCriteria, které jsme si v kapitole 3.4 více přiblížili. Nyní si můžeme uvést strukturu CvSVMParams, která obsahuje veškeré potřebné parametry pro trénování SVM. Jako první máme parametr svm_type, který obsahuje 5 možných hodnot, a to C_SVC, NU_SVC, ONE_CLASS, EPS_SVR a NU_SVR. V našich experimentech se nejlépe osvědčilo nastavení parametru na C_SVC, které si tedy více popíšeme. Mějme trénovací vektory $\mathbf{x}_i \in \mathbb{R}^n$, kde $n = 1, \ldots, l$ ve dvou třídách a vektor $y \in \mathbb{R}^l$, kde $y_i \in \{1, -1\}$ a optimalizační problém, který řeší C-SVC, tedy

$$\min_{\mathbf{w},b,\xi} \qquad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i, \qquad \text{kde}\, y_i\left(\mathbf{w}^T\phi\left(\mathbf{x}_i\right) + b\right) \geq 1 - \xi_i, \xi_i \geq 0,\ i = 1, \ldots, l, \quad (53)$$

kde $\phi\left(\mathbf{x}_i\right)$ mapuje vektor $\mathbf{x}_i$ do vícerozměrného prostoru a $C > 0$ je regularizační parametr.

Vzhledem k možnému vysokému počtu prostorů vektoru $\mathbf{w}$ většinou řešíme duální problém

$$\min_{\alpha} \qquad \frac{1}{2}\alpha^T Q\alpha - 1^T\alpha, \qquad \text{kde}\, y^T\alpha = 0,\ 0 \leq \alpha_i \leq C,\ i = 1, \ldots, l, \quad (54)$$

kde $\mathbf{1} = [1, \ldots, 1]^T$ je jedničkový vektor, Q je pozitivní semidefinitní matice $l \times l$, $Q_{ij} \equiv y_i y_j K\left(\mathbf{x}_i, \mathbf{x}_j\right)$ a $K\left(\mathbf{x}_i, \mathbf{x}_j\right) \equiv \phi\left(\mathbf{x}_i\right)^T \phi\left(\mathbf{x}_j\right)$ je kernel funkce. Jakmile máme tento problém vyřešen, můžeme získat

$$\mathbf{w} \quad = \quad \sum_{i=1}^{l} y_i \alpha_i \phi\left(\mathbf{x}_i\right) \quad (55)$$

a rozhodovací funkce pak bude

$$\text{sgn}\left(\mathbf{w}^T\phi\left(\mathbf{x}\right) + b\right) \quad = \quad \text{sgn}\left(\sum_{i=1}^{l} y_i \alpha_i K\left(\mathbf{x}_i, \mathbf{x}\right) + b\right). \quad (56)$$

V modelu predikce se tak ukládá $y_i\alpha_i \forall i$, b, názvy označení (± 1), podpůrné vektory a parametry kernelu.
Podrobnější popis k ostatním hodnotám parametru svm_type nalezneme v [18].

Jako další parametr ke struktuře CvSVMParams si uveďme kernel_type, tedy typ kernelu. Ten může nabývat opět několika hodnot a jako první si můžeme zmínit hodnotu LINEAR. LINEAR je lineární kernel s nejrychlejším výpočtem ($K\left(\mathbf{x}_i, \mathbf{x}_j\right) = \mathbf{x}_i^T\mathbf{x}_j$). Jako nejlepší volba se pro

naše experimenty ukázala být hodnota RBF (Radial basis function), který můžeme vypočíst jako $K(\mathbf{x}_i, \mathbf{x}_j) = e^{\gamma||\mathbf{x}_i - \mathbf{x}_j||^2}, \gamma > 0$. Poslední dvě možností jsou polynomický kernel, tedy POLY, a SIGMOID kernel, které se v našich experimentech ale neosvědčily.

Nyní si můžeme popsat ke struktuře CvSVMParams doplňující parametry k dříve zmíněným. Pro nastavení kernelu se jedná o parametr degree pro kernel funkci POLY, parametr gamma (γ, k funkcím POLY, RBF a SIGMOID) a parametr coef0 k funkcím POLY a SIGMOID. Pro nastavení SVM se jedná o C, tedy parametr C optimalizačního problému (pro C_SVC, EPS_SVR a NU_SVR), parametr nu, tedy parametr ν (pro NU_SVC, ONE_CLASS a NU_SVR), parametr p, tedy ϵ (pro EPS_SVR), parametr class_weights jsou volitelné váhy pro C_SVC pro konkrétní třídy (class_weights $_i \cdot C$ ovlivňující chybnou klasifikaci tříd) a jako poslední parametr si konečně můžeme uvést term_crit , který jsme si podrobněji uvedli v kapitole 3.4.

```
CvSVMParams params;
   params.svm_type = CvSVM::hodnota1;
   params.kernel_type = CvSVM::hodnota2;
   params.degree = hodnota3;
   params.gamma = hodnota4;
   params.coef0 = hodnota5;
   params.C = hodnota6;
   params.nu = hodnota7;
   params.p = hodnota8;
   params.class_weights = hodnota9;
   params.term_crit = crit;
```

Ukázka 3: Nastavení parametrů pro trénovaní rozhodovacího stromu.

Praktická ukázka implementace SVM pomocí OpenCV je součástí programu této práce a její výsledky můžeme vidět v kapitole 6.

5 Realizace rozhodovacích stromů a SVM

Nyní si můžeme více popsat vlastní implementační část naší diplomové práce. U implementace jsme zvolili vývojové prostředí MS Visual Studio 2013, programovací jazyk $C++$ a OpenCV open source knihovnu pro zpracování obrazu. Jelikož cílem této diplomové práce nebyla samotná implementace rozhodovacích stromů a SVM, nýbrž jejich teoretický rozbor a ověření v praxi, využili jsme již hotových knihoven OpenCV (ve verzi 2.4.12) [16]. Ze stejného důvodu postačí pro testovací účely také konzolová aplikace. Vhodná testovací a trénovací data byla poskytnuta skupinou Media Research Lab [17], díky čemuž mohlo být praktické porovnání provedeno na užitečných snímcích z praxe pro katedru.

Zopakujme si, že rozhodovací strom v OpenCV 2.4.12 je reprezentován třídou CvDTree, zatímco SVM je reprezentován pomocí třídy CvSVM. Obě tyto třídy byly použity pro natrénování daného klasifikátoru (a poté samozřejmě i k jeho testování). K výpočtu vhodných příznaků na naší vstupní sadě dat (trénovací i testovací) byla použita metoda HOG, která je reprezentována třídou HOGDescriptor.

5.1 Vstupy a výstupy programu

Jako vstup do našeho programu máme sadu RGB obrazů parkovišť snímaných z kamery v areálu VŠB. Každý z těchto obrazů parkoviště obsahuje 56 parkovacích míst. Pro trénování klasifikátoru jsme použili 101 obrazů parkoviště, tedy 5656 parkovacích míst (2968 plně prázdných a 2688 plně obsazených míst) a pro trénování klasifikátoru jsme použili 24 obrazů parkoviště, tedy 1344 různě obsazených míst pro následné otestování našeho zvoleného klasifikátoru. Ukázku vstupních obrazů pro trénování klasifikátoru můžeme vidět na obr. 16 a ukázku vstupních obrazů pro testování klasifikátoru můžeme vidět na obr. 17.

Obrázek 16: Ukázka vstupních obrazů pro trénování klasifikátoru

Důležitá informace pro zpracování těchto vstupních dat, ať už trénovacích či testovacích obrazů, je rozložení parkovacích míst na parkovišti, tedy jeho geometrie pro následné rozdělení parkoviště na jednotlivá parkovací místa, která jsou následným vstupem pro HOG algoritmus

Obrázek 17: Ukázka vstupních obrazů pro testování klasifikátoru

pro výpočet příznaků. Tuto informaci máme uloženou v textovém souboru (input_geometry.txt), ve kterém jsou uloženy jednotlivé souřadnice x a y $2D$ obrazu parkovišť. Tento soubor je důležitým vstupem jak do trénovací části programu, tak do jeho testovací části, jelikož v každé z nich není vstupem konkrétní parkovací místo, nýbrž celé parkoviště, ve kterém pak zkoumáme jednotlivá parkovací místa. Na obr. 18 můžeme vidět ukázku několika pozitivních (obsazených) a negativních (neobsazených) dat pro naše klasifikátory, jinými slovy již separovaná konkrétní parkovací místa z parkoviště ke klasifikaci.

Obrázek 18: Ukázka pozitivních a negativních dat (obsazených a neobsazených parkovacích míst)

Trénovací a testovací sadu obrazů máme rozdělenou ve dvou adresářích a program dostane jako vstup soubor se seznamem cest ke každé z těchto sad, tedy se seznamem vstupních obrazů testovacích i trénovacích. Trénovací sadu máme navíc ještě rozdělenu na dvě části a to z důvodu rozlišení prázdných a plně obsazených parkovišť za účelem naučení klasifikátoru správné obsazenosti parkovacího místa.

Výstupem, který mají obě metody společný, je textový soubor obsahující vypočítané příznaky (train_vector .txt) námi zvolenou metodou HOG, které jsou následně vstupem pro natré-

nování jednotlivých klasifikátorů. Výstupem natrénování klasifikátoru je .xml soubor, který je poté zpětně vstupem pro následné jeho otestování.

Pro zhodnocení účinnosti dané metody využíváme dva soubory, a to detectorOutput.txt a groundtruth.txt. První soubor obsahuje získané informace o obsazenosti parkovacích míst a druhý obsahuje správné informace o obsazenosti parkovacích míst, což nám umožňuje výsledné zhodnocení správnosti metody. Za zmínku stojí také náš soubor Results_of_methods.xlsx obsahující záznamy o účinnosti jednotlivých metod při experimentech se zvolenými parametry.

Jelikož jsme si již nadefinovali potřebné vstupy a výstupy pro náš program, můžeme přejít k samotnému trénování a testování klasifikátorů.

5.2 Trénování a testování SVM

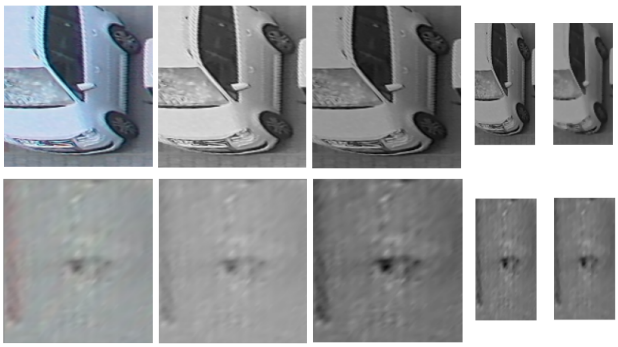
Jako první si uveďme trénování a testování SVM klasifikátoru. Než-li se pustíme do první části práce s klasifikátorem - jeho trénování - musíme si vstupní obrazy nejdříve předpřipravit. Úpravy, které jsme použili na naše vstupní data (jednotlivá parkovací místa), jsou

- změna barevného schématu (cvtColor) z RGB na stupně šedi,
- normalizace (normalize),
- změna velikosti (resize) a
- rozostření (medianBlur).

Tyto úpravy byly provedeny pro zefektivnění a urychlení výpočtu příznaků pomocí metody HOG. Jakmile máme takto upraveny vstupní obrazy, tak můžeme přejít k samotnému výpočtu příznaků pomocí HOG metody compute. Takto postupně vypočítáme pro každý vstupní obraz vektor příznaků, ke kterému ale nesmíme zapomenout přidat důležitou informaci o obsazenosti parkovacího místa (0 pro prázdné, 1 pro obsazené), kterou přidáme vždy na konec vektoru příznaků, a celkovou informaci uložíme do souboru pro trénování SVM.

Jakmile máme vytvořen soubor příznaků pro všechny vstupní trénovací obrazy, tak můžeme přejít k samotnému natrénování SVM klasifikátoru. Jako vstup využijeme soubor získaný z předešlého kroku obsahující vektory příznaků a jejich klasifikaci do dvou tříd (obsazeno a neobsazeno). Z tohoto souboru extrahujeme jednotlivé vektory a uložíme jej do dvou polí (s daty a jejich odpověďmi).

Před samotným trénováním klasifikátoru je ještě nutno nastavit jeho parametry. Nastavení parametrů pro SVM má dvě části - první část se týká nastavení pravidel zastavení iterativního procesu a druhá se týká samotného klasifikátoru SVM. Pro první nastavení jsme se rozhodli



Obrázek 19: Ukázka jednotlivých úprav vstupních obrazů

vyzkoušet dva typy - CV_TERMCRIT_ITER a CV_TERMCRIT_EPS a k nim odpovídající hodnoty proměnných max_iter a epsilon.

K samotným parametrům SVM jsme se rozhodli otestovat hlavně C_SVC typ SVM a k němu jednotlivé typy jader, které OpenCV nabízí a možné další parametry, které ovlivňují výslednou přesnost SVM klasifikátoru (více můžeme vidět v kapitole 6). Jakmile máme nastaveny všechny potřebné parametry tak se můžeme pustit do samotného natrénování SVM pomocí funkce train. Výsledky natrénování jsou poté uloženy do souboru svmHOG.xml a připraveny k dalšímu použití v rámci testovací fáze programu.

V testovací fázi SVM musíme opět vypočítávat jednotlivé vektory příznaků vstupních obrazů pomocí metody HOG se stejným nastavením jako při trénování a pomocí funkce predict můžeme spustit samotné testování funkčnosti našeho SVM s vypočtenými HOG příznaky. Metoda bude na základě souboru xml získaného z trénování klasifikátoru klasifikovat nová, dříve neviděná data a určovat, zda se na obrazu vyskytuje či nevyskytuje automobil. Tyto výsledky se poté porovnají s pravdivými klasifikacemi uloženými v souboru, porovnají se tedy dříve zmíněné soubory detectorOutput.txt a groundtruth.txt a pomocí tohoto porovnání můžeme pak zjistit úspěšnost klasifikátoru. Výsledky metody, tedy její úspěšnost, nakonec znázorníme graficky pomocí barevných rozlišení „vložených“ do testovacího obrazu.

5.3 Trénování a testování rozhodovacího stromu

Postup pro trénování a testování rozhodovacích stromů je velmi podobný postupu pro SVM klasifikátor. Před začátkem trénování rozhodovacího stromu si znovu předpřipravíme vstupní obrazy, a to opět

- změnou barevného schématu (cvtColor) z RGB na stupně šedi,
- normalizací (normalize),

- změnou velikosti (resize) a
- rozostřením (medianBlur)

z důvodu zefektivnění a urychlení výpočtů pomocí HOG. Výpočet příznaků pomocí HOG probíhá opět pomocí metody compute a následně k souboru příznaků ještě přiřadíme správné odpovídající klasifikace (obsazenosti či neobsazenosti parkovacího místa). Po vytvoření všech vektorů příznaků můžeme přejít k trénování rozhodovacího stromu.

Vstupem pro trénování klasifikátoru je opět vektor příznaků vypočtený pomocí metody HOG, ze kterého opět extrahujeme potřebná data (příznaky a obsazenost parkovacího místa). Pro trénování rozhodovacího stromu nenastavujeme parametry iterativního zastavení procesu, tato možnost je pouze u *náhodných lesů* (*random forest*) a nastavujeme proto pouze samotné parametry rozhodovacího stromu jako takového. Otestovat jsme se rozhodli hlavně různé nastavení hloubky stromu, jelikož ta by výslednou přesnost měla ovlivňovat nejvíce, a také počet vzorků v uzlech, které by měly mít taktéž nemalý vliv na výslednou přesnost rozhodovacího stromu. Jednotlivé experimenty s jejich výsledky poté můžeme vidět v kapitole 6. Po nastavení potřebných parametrů opět použijeme funkci train , a to tentokrát pro trénování našeho rozhodovacího stromu. Výsledky jsou poté uloženy v souboru decTreeHOG.xml.

Jakmile jsme dokončili trénování rozhodovacího stromu tak můžeme přejít na jeho otestování. Opět znovu vypočítáváme vektory příznaků pro vstupní obrazy (testovací sadu) pomocí HOG se stejným nastavením abychom získali správné a nezkreslené výsledky. Pro rozhodovací stromy využijeme funkci predict , která bude klasifikovat nová, dříve neviděná testovací data, pro která klasifikátor nemá předem určen jejich správnou klasifikaci. Výsledky opět porovnáme pomocí obsahů dvou souborů, a to detectorOutput.txt a groundtruth.txt , díky čemuž můžeme poté získat výslednou přesnost rozhodovacího stromu pro dané nastavení jeho parametrů a zaznamenat je graficky do výsledných testovacích obrazů.

6 Experimenty

Cílem praktické části naší práce bylo otestování užitečnosti rozhodovacích stromů a SVM na detekci obsazenosti parkovacích míst na parkovišti. Můžeme si zopakovat, že pro výpočet příznaků jsme použili metodu HOG.

Úspěšnost jak rozhodovacího stromu, tak SVM, jsme měřili jako

$$\text{presnost} = \frac{\text{spravne pozitivni} + \text{spravne negativni}}{\text{spravne pozitivni} + \text{spravne negativni} + \text{chybne pozitivni} + \text{chybne negativni}}, \tag{57}$$

kde správně klasifikovaná data byla ta, co správně klasifikují obsazenost parkovacího místa, kdežto chybně klasifikovaná data byla ta, která prázdné místo označí jako obsazené a naopak, tedy chybně pozitivní/negativní klasifikace.

Parametry pro HOG deskriptor jsme pro otestování na rozhodovacích stromech a SVM nechali s defaultním nastavením s výjimkou velikosti okna, které jsme nastavili na $(16, 16)$. Důležitá je i podmínka pro velikost okna HOG v souvislosti k rozměrům vstupních obrazů, jelikož musí být shodné.

```
Size win_size=Size(16, 16),
Size block_size=Size(16, 16),
Size block_stride=Size(8, 8),
Size cell_size=Size(8, 8),
int nbins=9,
double win_sigma=DEFAULT_WIN_SIGMA,
double threshold_L2hys=0.2,
bool gamma_correction=true,
int nlevels=DEFAULT_NLEVELS
```

Ukázka 4: Nastavení parametrů pro HOG.

Nastavení pro rozhodovací stromy a SVM bylo různé a uvedeme si je v následujících podkapitolách, ve kterých si zhodnotíme jednotlivé experimenty, tedy jejich výslednou prediktivní přesnost.

6.1 Experimenty s rozhodovacími stromy

Podrobný popis k jednotlivým parametrům rozhodovacích stromů jsme si popsali v kapitole 3.4 a v těchto kapitolách tedy přejdeme k samotným experimentům s nimi.

Pro první experiment jsme nastavili konstantní hodnoty pro všechny parametry kromě maximální hloubky stromu, kterou jsme se rozhodli otestovat jako první. Výsledky spolu s hodnotami konkrétních nastaveních můžeme vidět na obr. 20. Nejlepší přesnosti $0,709077381$ rozhodovacího stromu podle rovnice (57) jsme dosáhli u hodnoty 7 pro maximální hloubku rozhodovacího stromu. Vyšší hodnota než 10 maximální hloubky stromu nám již žádné zlepšení predikce stromu nepřinesla.

max_depth	1	2	3	4	5	6	**7**	8	9	10	11	15	30
min_sample_count	10	10	10	10	10	10	**10**	10	10	10	10	10	10
use_surrogates	true	true	true	true	true	true	**true**	true	true	true	true	true	true
cv_folds	0	0	0	0	0	0	**0**	0	0	0	0	0	0
use_1se_rule	true	true	true	true	true	true	**true**	true	true	true	true	true	true
truncate_pruned_tree	true	true	true	true	true	true	**true**	true	true	true	true	true	true
false positive	372	408	435	353	373	350	**343**	341	341	338	338	338	338
false negative	103	68	47	44	36	42	**48**	51	52	54	54	54	54
true positive	322	357	378	381	389	383	**377**	374	373	371	371	371	371
true negative	547	511	484	566	546	569	**576**	578	578	581	581	581	581
Accuracy	0,646577	0,645833	0,641369	0,704613	0,695685	0,708333	**0,709077**	0,708333	0,707589	0,708333	0,708333	0,708333	0,708333
Max accuracy:	**0,709077**												

Obrázek 20: Rozhodovací strom: Experiment s hloubkou stromu, nejlepší přesnost $0,709077381$

Jakmile jsme zjistili nejvhodnější hloubku stromu pro naše stávající nastavení, rozhodli jsme se otestovat vliv minimálního počtu vzorků v uzlu stromu pro nejlepší hloubku stromu z předchozího experimentu, tedy pro hodnotou 7. Výsledky můžeme vidět na obr. 21 a obr. 22.

max_depth	7	7	7	7	7	7	7	**7**	7	7	7	7	7
min_sample_count	1	5	10	15	20	30	40	**50**	60	70	80	90	100
use_surrogates	true	true	true	true	true	true	true	**true**	true	true	true	true	true
cv_folds	0	0	0	0	0	0	0	**0**	0	0	0	0	0
use_1se_rule	true	true	true	true	true	true	true	**true**	true	true	true	true	true
truncate_pruned_tree	true	true	true	true	true	true	true	**true**	true	true	true	true	true
false positive	368	368	343	346	346	335	335	**330**	335	342	354	354	347
false negative	47	47	48	48	48	49	49	**50**	47	47	47	47	46
true positive	378	378	377	377	377	376	376	**375**	378	378	378	378	379
true negative	551	551	576	573	573	584	584	**589**	584	577	565	565	572
Accuracy	0,69122	0,69122	0,709077	0,706845	0,706845	0,714286	0,714286	**0,717262**	0,715774	0,710565	0,701637	0,701637	0,707589
Max accuracy:	**0,717262**												

Obrázek 21: Rozhodovací strom: Experiment s minimálním počtem vzorků, výsledná přesnost $0,717261905$

V první části jsme vyzkoušeli velké rozmezí hodnot pro zjištění přibližných prediktivních přesností pro jednotlivé minimální počty vzorků v uzlech, a poté jsme si z nich vybrali nejlepší výsledek, pro který jsme udělali jemnější experiment, tedy průzkum okolních hodnot po jednotkách. U nejlepšího výsledku se nám přesnost rozhodovacího stromu zlepšila o $0,008184524$ a výsledná přesnost po nastavení minimálního počtu vzorků byla tedy $0,717261905$, a to pro několik hodnot okolo hodnoty 50 včetně, my jsme si nicméně pro další experimenty vybrali právě tuto hodnotu.

Mírný vliv měl na náš rozhodovací strom parametr cv_folds , při jehož použití jsme ale dosáhli nižších hodnot než bez něj. Můžeme tedy říci, že prořezání stromu nemělo kladný efekt na náš

max_depth	7	7	7	7	7	**7**	7	7	7	7	7
min_sample_count	45	46	47	48	49	**50**	51	52	53	54	55
use_surrogates	true	true	true	true	true	**true**	true	true	true	true	true
cv_folds	0	0	0	0	0	**0**	0	0	0	0	0
use_1se_rule	true	true	true	true	true	**true**	true	true	true	true	true
truncate_pruned_tree	true	true	true	true	true	**true**	true	true	true	true	true
false positive	335	335	335	335	330	**330**	330	330	330	330	335
false negative	49	49	49	49	50	**50**	50	50	50	50	47
true positive	376	376	376	376	375	**375**	375	375	375	375	378
true negative	584	584	584	584	589	**589**	589	589	589	589	584
Accuracy	0,714286	0,714286	0,714286	0,714286	0,717262	**0,717262**	0,717262	0,717262	0,717262	0,717262	0,715774
Max accuracy:	**0,717262**										

Obrázek 22: Rozhodovací strom: Upřesnění výsledné hodnoty minimálního počtu vzorků, výsledná přesnost $0,717261905$

rozhodovací strom obsahující pouze dvě třídy pro klasifikaci. Výsledky můžeme opět vidět na obr. 23.

max_depth	**7**	7	7	7	7	7	7	7
min_sample_count	**50**	50	50	50	50	50	50	50
use_surrogates	**true**	true	true	true	true	true	true	true
cv_folds	**0**	0,5	1	1,5	2	3	5	10
use_1se_rule	**true**	true	true	true	true	true	true	true
truncate_pruned_tree	**true**	true	true	true	true	true	true	true
false positive	**330**	330	330	330	466	466	466	466
false negative	**50**	50	50	50	50	50	50	50
true positive	**375**	375	375	375	375	375	375	375
true negative	**589**	589	589	589	453	453	453	453
Accuracy	**0,717262**	0,717262	0,717262	0,717262	0,616071	0,616071	0,616071	0,616071
Max accuracy:	**0,717262**							

Obrázek 23: Rozhodovací strom: Experiment s možností prořezávání stromu

Jestliže jsme prořezávání stromu nepoužili, pak výsledná přesnost byla $0,717261905$, při jeho použití se zhoršila o celých $0,101190476$, konkrétně na $0,616071429$.

Jako další experiment jsme otestovali maximální hloubku stromu pro novou hodnotu minimálního počtu vzorků v uzlu stromu pro případné následující dělení, tedy pro hodnotu 50 získanou z předešlého experimentu. Výsledky můžeme vidět na obr. 24.

Díky tomuto experimentu jsme si ověřili, že námi zvolená hloubka rozhodovacího stromu 7 i minimální počet vzorků v uzlu stromu byl správný, jelikož již nedošlo ke zlepšení nejlepší prediktivní přesnosti rozhodovacího stromu ani při jiné hloubce stromu.

Zbývající parametry neměly na naše experimenty vliv a jejich výsledky můžeme vidět v přiloženém disku v souboru se záznamy z experimentů našich metod.

max_depth	1	2	3	4	5	6	**7**	8	9	10	11	12	15	30
min_sample_count	50	50	50	50	50	50	**50**	50	50	50	50	50	50	51
use_surrogates	true	true	true	true	true	true	**true**	true	true	true	true	true	true	true
cv_folds	0	0	0	0	0	0	**0**	0	0	0	0	0	0	1
use_1se_rule	true	true	true	true	true	true	**true**	true	true	true	true	true	true	true
truncate_pruned_tree	true	true	true	true	true	true	**true**	true	true	true	true	true	true	true
false positive	372	408	435	353	365	341	**330**	328	328	327	327	327	327	327
false negative	103	68	47	44	37	44	**50**	53	54	55	55	55	55	55
true positive	322	357	378	381	388	381	**375**	372	371	370	370	370	370	370
true negative	547	511	484	566	554	578	**589**	591	591	592	592	592	592	592
Accuracy	0,646577	0,645833	0,641369	0,704613	0,700893	0,713542	**0,717262**	0,716518	0,715774	0,715774	0,715774	0,715774	0,715774	0,715774
Max accuracy:	**0,717262**													

Obrázek 24: Rozhodovací strom: Experiment s hloubkou stromu, nejlepší přesnost $0,717261905$

Nejlepší přesnost, jakou se nám povedlo pomocí metody HOG pro rozhodovací stromy v OpencCV získat, byla tedy $0,717261905$, kdy naše metoda klasifikovala správně 964 vzorů (tedy parkovacích míst a jejich obsazenost automobily) a chybně klasifikovala celých 380 vzorů testovacích dat.

6.2 Experimenty s SVM

Podrobný popis k jednotlivým parametrům SVM jsme si popsali v kapitole 4.4 a můžeme i zde přejít k samotným experimentům.

Pro otestování SVM jsme si vybrali a otestovali úspěšnost pro typ C−SVC, který jsme si podrobněji popsali v kapitole 4.4. Jako první jsme otestovali iterační kritérium pro zastavení iterativního procesu, tedy CV_TERMCRIT_ITER, a určili si konstanty pro všechny parametry kromě maximálního počtu iterací, který jsme otestovali jako první. Výsledky, stejně tak jako jednotlivé hodnoty různých nastavení, můžeme vidět na obr. 25 a 26.

epsilon	1	1	1	1	1	1	1	1	**1**	1	1	1	1	1	1
max_iter	1	2	3	4	5	10	20	30	**50**	75	100	125	150	175	200
type	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	**ITER**	ITER	ITER	ITER	ITER	ITER	ITER
svm_type	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	**C_SVC**	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC
kernel_type	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR	**LINEAR**	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR
gamma	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	**0,1**	0,1	0,1	0,1	0,1	0,1	0,1
C	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	**0,01**	0,01	0,01	0,01	0,01	0,01	0,01
nu	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	**0,1**	0,1	0,1	0,1	0,1	0,1	0,1
false positive	730	620	238	520	548	256	320	369	**284**	328	306	295	317	319	319
false negative	156	35	90	100	55	76	10	9	**10**	9	6	11	9	12	10
true positive	269	390	335	325	370	349	415	416	**415**	416	419	414	416	413	415
true negative	189	299	681	399	371	663	599	550	**635**	591	613	624	602	600	600
Accuracy	0,340774	0,512649	0,755952	0,53869	0,551339	0,752976	0,754464	0,71875	**0,78125**	0,749256	0,767857	0,772321	0,75744	0,75372	0,755208
Max accuracy:	**0,78125**														

Obrázek 25: SVM: Experiment s maximálním možným počtem iterací pro lineární jádro, výsledná přesnost $0,78125$

Podobně jako u rozhodovacích stromů jsme i zde nejprve testovali širší rozmezí hodnot (obr. 25), ve kterém jsme nalezli potenciální nejlepší prediktivní přesnost, a poté jsme si kolem v rozmezí této hodnoty provedli detailnější experimenty pro zjištění opravdové nejlepší prediktivní přesnosti SVM k danému nastavení (obr. 26).

V druhé části jsme tedy dosáhli přesnosti SVM 0,839285714 pro lineární jádro, a to konkrétně pro nastavení maximálního počtu iterací na 57. Již nyní můžeme vidět výrazně lepší výsledky prediktivní přesnosti oproti rozhodovacím stromům, a to i u hodnot s nižší než maximální prediktivní přesností.

epsilon	1	1	1	1	1	1	1	1	1	1	**1**	1	1	1
max_iter	35	40	45	50	51	52	53	54	55	56	**57**	58	59	60
type	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	**ITER**	ITER	ITER	ITER
svm_type	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	**C_SVC**	C_SVC	C_SVC	C_SVC
kernel_type	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR	**LINEAR**	LINEAR	LINEAR	LINEAR
gamma	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	**0,1**	0,1	0,1	0,1
C	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	**0,01**	0,01	0,01	0,01
nu	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	**0,1**	0,1	0,1	0,1
false positive	352	271	273	284	274	321	277	308	252	352	**202**	255	327	280
false negative	11	8	9	10	6	6	12	11	7	5	**14**	9	6	11
true positive	414	417	416	415	419	419	413	414	418	420	**411**	416	419	414
true negative	567	648	646	635	645	598	642	611	667	567	**717**	664	592	639
Accuracy	0,729911	0,792411	0,790179	0,78125	0,791667	0,756696	0,78497	0,762649	0,807292	0,734375	**0,839286**	0,803571	0,752232	0,783482
Max accuracy:	**0,839286**													

Obrázek 26: SVM: Upřesnění výsledné hodnoty maximálního možného počtu iterací, výsledná přesnost 0,839285714

V dalším experimentu jsme se rozhodli otestovat tento testovaný parametr znovu, nýbrž tentokrát nikoli pro lineární typ jádra, ale pro RBF, u kterého jsme očekávali lepší výsledky. Výsledky použití tohoto nastavení můžeme vidět na obr. 27. Naše očekávání nás nezklamalo a tento typ nám dal výrazně lepší prediktivní přednost SVM, a to 0,982886905, tedy o celých 0,14360119 více než u předchozího nastavení.

epsilon	1	1	1	1	1	1	1	1	1	1	1	1	1	**1**	1	1	1	1	1
max_iter	1	5	10	20	30	40	50	55	56	57	58	59	60	**61**	62	63	64	65	70
type	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	**ITER**	ITER	ITER	ITER	ITER	ITER
svm_type	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	**C_SVC**	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC
kernel_type	RBF	RBF	RBF	RBF	RBF	RBF	RBF	RBF	RBF	RBF	RBF	RBF	RBF	**RBF**	RBF	RBF	RBF	RBF	RBF
gamma	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	**0,1**	0,1	0,1	0,1	0,1	0,1
C	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	**0,01**	0,01	0,01	0,01	0,01	0,01
nu	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	**0,1**	0,1	0,1	0,1	0,1	0,1
false positive	9	486	236	102	57	45	15	16	13	14	15	21	15	**14**	16	16	14	14	16
false negative	416	14	10	11	9	8	11	10	10	10	10	10	9	**9**	10	10	10	10	10
true positive	9	411	415	414	416	417	414	415	415	415	415	415	416	**416**	415	415	415	415	415
true negative	910	433	683	817	862	874	904	903	906	905	904	898	904	**905**	903	903	905	905	903
Accuracy	0,68378	0,627976	0,816964	0,915923	0,950893	0,960565	0,980655	0,980655	0,982887	0,982143	0,981399	0,976935	0,982143	**0,982887**	0,980655	0,980655	0,982143	0,982143	0,980655
Max accuracy:	**0,982887**																		

Obrázek 27: SVM: Experiment s maximálním možným počtem iterací pro jádro RBF, výsledná přesnost 0,982886905

V rámci dalších experimentů jsme se rozhodli otestovat vliv parametru gamma na výslednou prediktivní přesnost. Výsledky, jakožto i jednotlivá nastavení, můžeme vidět na obr. 28. Touto modifikací se nám povedlo dosáhnout prediktivní přesnosti až 0,988095238, tedy o 0,005208333 více než v předchozím experimentu, můžeme tedy vidět, že již nedošlo k příliš velké změně v přesnosti klasifikátoru oproti předchozím experimentům.

Ukázku experimentu výsledků neiteračního kritéria pro zastavení iterativního procesu, tedy CV_TERMCRIT_EPS, můžeme vidět na obr. 29. Výsledná prediktivní přesnost SVM pro lineární

epsilon	1	1	1	1	1	**1**	1	1	1	1	1	1	1	1	1	1	1	1
max_iter	61	61	61	61	61	**61**	61	61	61	61	61	61	61	61	61	61	61	61
type	ITER	ITER	ITER	ITER	ITER	**ITER**	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER	ITER
svm_type	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	**C_SVC**	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC
kernel_type	RBF	RBF	RBF	RBF	RBF	**RBF**	RBF	RBF	RBF	RBF	RBF	RBF	RBF	RBF	RBF	RBF	RBF	RBF
gamma	0	0,001	0,01	0,05	0,1	**0,11**	0,12	0,13	0,14	0,15	0,16	0,17	0,18	0,19	0,2	0,3	0,4	0,5
C	0,01	0,01	0,01	0,01	0,01	**0,01**	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01
nu	0,1	0,1	0,1	0,1	0,1	**0,1**	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1
false positive	x	264	215	52	14	**8**	12	7	8	11	12	13	8	9	7	3	315	352
false negative	x	10	4	8	9	**8**	8	10	10	11	11	11	14	15	18	27	21	19
true positive		415	421	417	416	**417**	417	415	415	414	414	414	411	410	407	398	404	406
true negative		655	704	867	905	**911**	907	912	911	908	907	906	911	910	912	916	604	567
Accuracy		0,796131	0,837054	0,955357	0,982887	**0,988095**	0,985119	0,987351	0,986607	0,983631	0,982887	0,982143	0,983631	0,982143	0,981399	0,977679	0,75	0,723958
Max accuracy:	**0,988095**																	

Obrázek 28: SVM: Experiment s parametrem gamma, výsledná přesnost $0,988095238$

jádro byla $0,792410714$ a pro RBF $0,904761905$, tudíž ve srovnání s naším předchozím modelem se jednalo o mnohem horší výsledky.

epsilon	0	0,1	0,5	1	2
max_iter	1	1	1	1	1
type	EPS	EPS	EPS	EPS	EPS
svm_type	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC
kernel_type	RBF	RBF	RBF	RBF	RBF
gamma	0,1	0,1	0,1	0,1	0,1
C	0,01	0,01	0,01	0,01	0,01
nu	0,1	0,1	0,1	0,1	0,1
false positive	0	0	0	0	117
false negative	425	425	425	425	11
true positive	0	0	0	0	414
true negative	919	919	919	919	802
Accuracy	0,68378	0,68378	0,68378	0,68378	0,904762
Max accuracy:	**0,904762**				

epsilon	1	2	3	4	5
max_iter	1	1	1	1	1
type	EPS	EPS	EPS	EPS	EPS
svm_type	C_SVC	C_SVC	C_SVC	C_SVC	C_SVC
kernel_type	LINEAR	LINEAR	LINEAR	LINEAR	LINEAR
gamma	0,1	0,1	0,1	0,1	0,1
C	0,01	0,01	0,01	0,01	0,01
nu	0,1	0,1	0,1	0,1	0,1
false positive	427	271	919	919	919
false negative	10	8	0	0	0
true positive	415	417	425	425	425
true negative	492	648	0	0	0
Accuracy	0,674851	0,792411	0,31622	0,31622	0,31622
Max accuracy:	**0**				

Obrázek 29: SVM: Experiment s hodnotou ϵ

Použití iteračního kritéria pro zastavení iteračního procesu bylo tedy mnohem efektivnější, než použití zastavení iteračního procesu pomocí porovnávání přesnosti nižší než námi nadefinované pro klasifikátor.

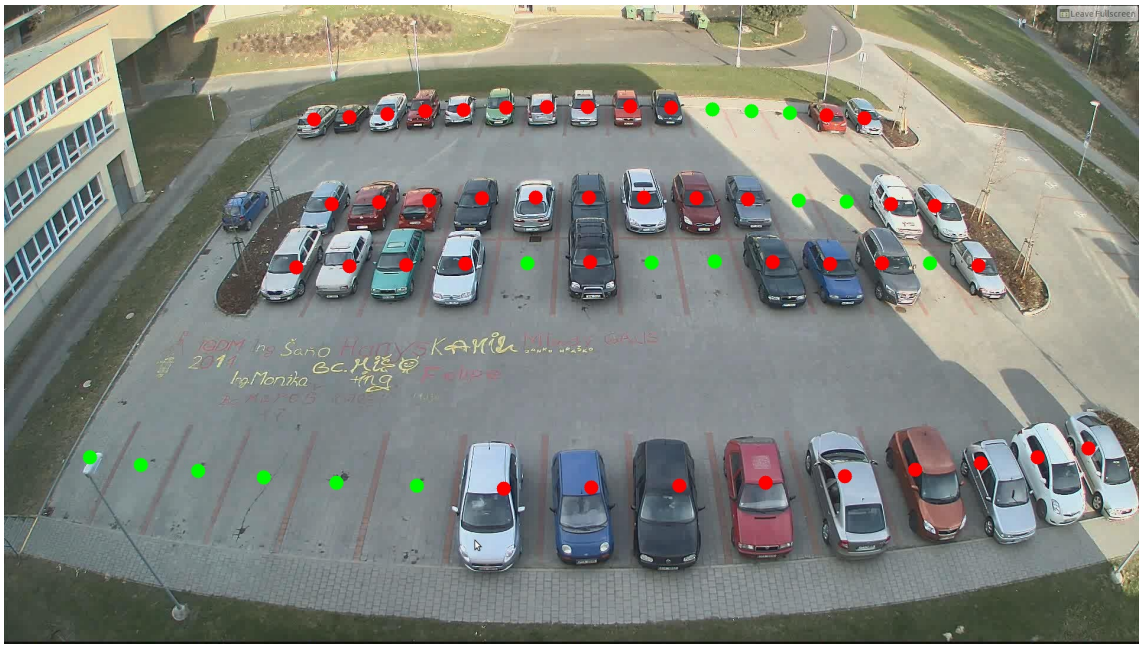
Nejlepší přesnost, které se nám povedlo pro SVM klasifikátor dosáhnout, byla tedy $0,988095238$, kdy nám klasifikátor klasifikoval chybně pouhých 16 vstupních obrazů a správně klasifikoval celých 1328 vstupních dat.

Více experimentů můžeme opět vidět na přiloženém disku v souboru se záznamy z experimentů našich metod, nicméně nám již žádné další nepomohly získat lepší prediktivní přesnost SVM.

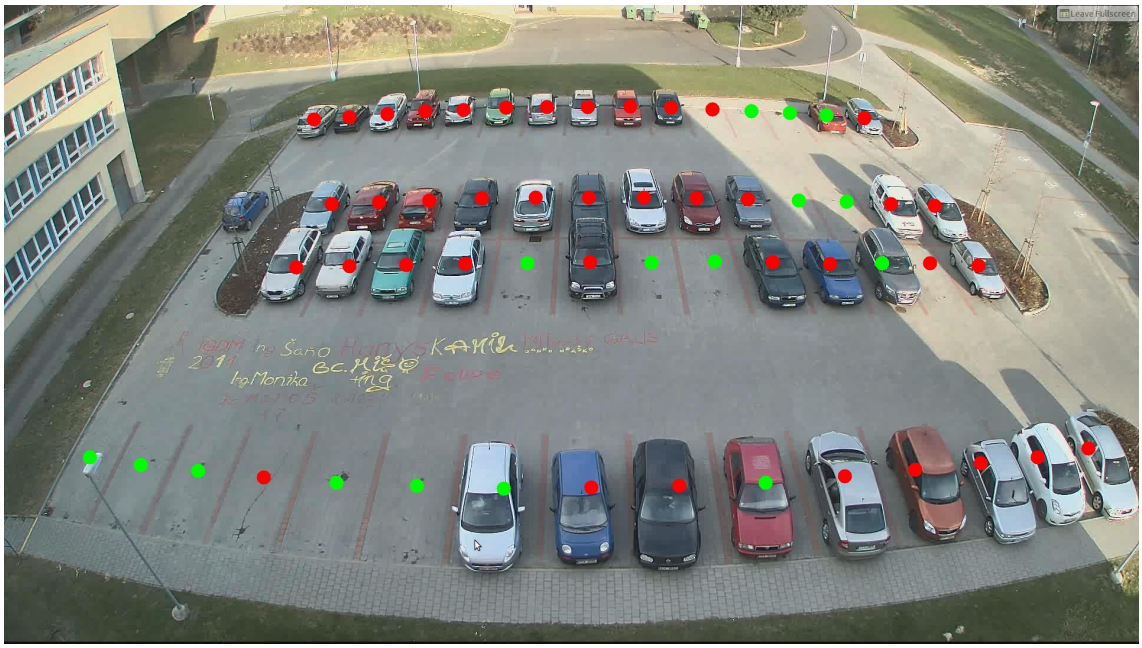
7 Závěr

V kapitolách 3 a 4 jsme si popsali detailnější teorii vybraných klasifikátorů, tedy rozhodovacích stromů a SVM. V kapitole 6 jsme zhodnotili užitečnost jednotlivých klasifikátorů pro metodu Histogram orientovaných gradientů, která má velmi dobré úspěchy u detekce osob a vyzkoušeli jsme jeho účinnost pro detekci automobilů, resp. detekci obsazenosti parkovacích míst na parkovišti.

Klasifikátor SVM vykazoval významně vyšší prediktivní přesnost oproti rozhodovacím stromům, které se příliš neosvědčily pro klasifikaci pomocí HOG na detekci parkovacích míst, a to tedy pouze pro dvě možné výsledné kategorie - obsazenost či neobsazenost parkovacího místa. Oproti tomu SVM ukázal výrazný předstih v prediktivní přesnosti oproti rozhodovacím stromům pro tento úkol s dvěma klasifikačními třídami a dosahoval prediktivní přesnosti až o hodnotách $0,988095238$ a tudíž je velmi účinným nástrojem pro zadaný úkol a jeho ukázku můžeme vidět na obr. 30. Ukázku výsledku klasifikace pomocí rozhodovacího stromu můžeme vidět na obr. 31.



Obrázek 30: Ukázka klasifikace pomocí SVM s prediktivní přesností $0,988095238$



Obrázek 31: Ukázka klasifikace pomocí Rozhodovacího stromu s prediktivní přesností $0,717261905$

Literatura

[1] Quinlan, J.R.: *Introduction of decision trees*, Machine Learning 1: 81-106, 1986

[2] Cortes, C., Vapnik V.: *Support-Vector Networks.* Machine Learning 20: 273–297, 1995

[3] Dalal, N.; Triggs, B.: *Histograms of Oriented Gradients for Human Detection.* In International Conference on Computer Vision Pattern Recognition, ročník 2, editace C. Schmid; S. Soatto; C. Tomasi, INRIA Rhône-Alpes, ZIRST-655, av. de l'Europe, Montbonnot-38334, June 2005, s. 886–893.

[4] Hunt, E.B., Marin, J., Stone, P.J. (1966). *Experiments in induction.* New York: Academic Press.

[5] Quinlan, J.R. (1979). *Discovering rules by induction from large collections of examples. In D. Michie (Ed.), Expert systems in the micro electronic age.* Edinburgh University Press.

[6] Quinlan, J.R. (1983). *Learning efficient classification procedures and their application to chess endgames. In R.S. Michalski, J.G. Carbonell T.M. Mitchell, (Eds.), Machine learning: An artificial intelligence approach.* Palo Alto: Tioga Publishing Company.

[7] Patterson, A., Niblett, T. (1983). *ACLS user manual.* Glasgow: Intelligent Terminals Ltd.

[8] Kononenko, I., Bratko, I., Roskar, E. (1984). *Experiments in automatic learning of medical diagnostic rules (Technical report).* Jozef Stefan Institute, Ljubljana, Yugoslavia.

[9] Eduard Sojka, Jan Gaura, Michal Krumnikl: *Matematické základy digitálního zpracování obrazu* (2011)

[10] Komprdová, K.: *Rozhodovací stromy a lesy* (2012)

[11] Rosenblatt, F.: *Principles ofNeurodynamics* (1962)

[12] Boser, B.E., Guyon, I., Vapnik, V.N. (1992). *A training algorithm for optimal margin classifiers. In Proceedings of the Fifth Annual Workshop of Computational Learning Theory*, 5, 144-152, Pittsburgh, ACM.

[13] Anderson, T.W., Bahadur, R.R. (1966). *Classification into two multivariate normal distributions with different covariance matrices.* Ann. Math. Stat., 33:420-431.

[14] Courant, R., Hilbert, D. (1953). *Methods of Mathematical Physics*, Interscience, New York.

[15] Fisher, R.A. (1936). *The use of multiple measurements in taxonomic problems.* Ann. Eugenics, 7:111-132.

[16] *Open Source Computer Vision Library*, http://opencv.org/

[17] *Media Research Lab (MRL)*, http://mrl.cs.vsb.cz/

[18] Chih-Chung Chang and Chih-Jen Lin, *A Library for Support Vector Machines*, http://www.csie.ntu.edu.tw/

A Obsah přiloženého disku

- text diplomové práce
- zdrojové kódy